Humza Salman, mhs180007

Section 6.9: Degree

In [121…

```
print('HIGHLIGHTED QUESTION - Identify an example type of network of when it is a good
print()
print('It would be good to be a high-degree node in a social-network like LinkedIn bec
```

```
HIGHLIGHTED QUESTION - Identify an example type of network of when it is a good thing
to be a high degree node and an example of when it is a bad thing to be a high degree
node in a network.

It would be good to be a high-degree node in a social-network like LinkedIn because t
hen you will have a lot of connections and a lot more opportunities in terms of your
career. It would be a bad thing to be a high degree node in a airport network because
it would result in higher connections which means a lot of wait times and delays.
```

In [122…

```
print('HIGHLIGHTED QUESTION - Describe or draw an example of a network in which a part
print()
print('Say we have a small neighborhood/town. Then a power grid network would be a goo
```

```
HIGHLIGHTED QUESTION - Describe or draw an example of a network in which a particular
node has very few connections, but it could be argued that it is a very important nod
e. Justify your reasoning.

Say we have a small neighborhood/town. Then a power grid network would be a good exam
ple of a network in which a node may have very few connections, but still be importan
t. For example, the power plant node is very important since it would supply electric
ity to nearby houses.
```

In [123…

```
print('HIGHLIGHTED QUESTION - Explain what equation (6.25) in Newman actually means -
print()
print('k_i^in is calculating the in-degree of node i by summing up the number of edges
```

```
HIGHLIGHTED QUESTION - Explain what equation (6.25) in Newman actually means - i.e.,
what is a simpler way to explain what the two summations mean?

k_i^in is calculating the in-degree of node i by summing up the number of edges going
from node j to node i. Similarly, k_j^out is calculating the out-degree of node j by
summing up the number of edges going from node j to node i.
```

Section 6.10-6.11: Paths & Components

In [124…

```
import numpy as np
```

In [125…

```
print('HIGHLIGHTED QUESTION - write down the adjacency matrix, A')
A = [[0,0,0,0,0],
     [1,0,0,0,0],
     [1,1,0,0,0],
     [0,1,1,0,0],
     [0,0,1,1,0]]
print(np.array(A))
```

```
HIGHLIGHTED QUESTION - write down the adjacency matrix, A
[[0 0 0 0 0]
 [1 0 0 0 0]
 [1 1 0 0 0]
 [0 1 1 0 0]
 [0 0 1 1 0]]
```

In [126…

```python
print('HIGHLIGHTED QUESTION - write down the adjacency matrix twice and multiply them
A2 = np.dot(A,A)
print(A2)
```

```
HIGHLIGHTED QUESTION - write down the adjacency matrix twice and multiply them to get
A^2.
[[0 0 0 0 0]
 [0 0 0 0 0]
 [1 0 0 0 0]
 [2 1 0 0 0]
 [1 2 1 0 0]]
```

In [127…

```python
print('HIGHLIGHTED QUESTION - Using equation (6.29) indicate what each nonzero element
# for i in range(len(A)):
#     for j in range(len(A[i])):
#         print(i*j)

print('Each non-zero element of A^2 represents the total number of paths of length 2 f
```

```
HIGHLIGHTED QUESTION - Using equation (6.29) indicate what each nonzero element of A^
2 means
Each non-zero element of A^2 represents the total number of paths of length 2 from no
de j to node i.
```

In [128…

```python
print('HIGHLIGHTED QUESTION - Write down the path(s) that correspond to these nonzero
print('Paths of length two to node 0:')
print('\tNone')
print('Paths of length two to node 1:')
print('\tNone')
print('Paths of length two to node 2:')
print('\t0 -> 1 -> 2')
print('Paths of length two to node 3:')
print('\t0 -> 1 -> 3')
print('\t0 -> 2 -> 3')
print('\t1 -> 2 -> 3')
print('Paths of length two to node 4:')
print('\t0 -> 2 -> 4')
print('\t1 -> 2 -> 4')
print('\t1 -> 3 -> 4')
print('\t2 -> 3 -> 4')
```

```
HIGHLIGHTED QUESTION - Write down the path(s) that correspond to these nonzero elemen
ts: e.g., 1 → 3 → 4.
Paths of length two to node 0:
        None
Paths of length two to node 1:
        None
Paths of length two to node 2:
        0 -> 1 -> 2
Paths of length two to node 3:
        0 -> 1 -> 3
        0 -> 2 -> 3
        1 -> 2 -> 3
Paths of length two to node 4:
        0 -> 2 -> 4
        1 -> 2 -> 4
        1 -> 3 -> 4
        2 -> 3 -> 4
```

In [ ]:

In [129…

```python
print('HIGHLIGHTED QUESTION - Multiply Ax, A^2x=A(Ax), and A^3x=A(A(Ax)). What do thes
x = [[1],[0],[0],[0],[0]]
A = [[0,0,0,0,0],
     [1,0,0,0,0],
     [1,1,0,0,0],
     [0,1,1,0,0],
     [0,0,1,1,0]]

Ax = np.dot(A,x)
A2x = np.dot(A, Ax)
A3x = np.dot(A, A2x)
# print(np.array(A))
# print(np.array(x))
print('Ax:')
print(Ax)

print('A2x:')
print(A2x)

print('A3x:')
print(A3x)

print('The resulting column vector represents the number of length 1, 2, or 3 paths fo
```

```
HIGHLIGHTED QUESTION - Multiply Ax, A^2x=A(Ax), and A^3x=A(A(Ax)). What do these resu
lting column vectors represent
Ax:
[[0]
 [1]
 [1]
 [0]
 [0]]
A2x:
[[0]
 [0]
 [1]
 [2]
 [1]]
A3x:
[[0]
 [0]
 [0]
 [1]
 [3]]
The resulting column vector represents the number of length 1, 2, or 3 paths for Ax,
A^2x, or A^3x, respectively from node 0 to any other node i
```

In [ ]:

```

```

In [130…

```
print('HIGHLIGHTED QUESTION - Now do the same for x=[0,0,1,0,0]T. Does your observatic
x = [[0],[0],[1],[0],[0]]
A = [[0,0,0,0,0],
     [1,0,0,0,0],
     [1,1,0,0,0],
     [0,1,1,0,0],
     [0,0,1,1,0]]

Ax = np.dot(A,x)
A2x = np.dot(A, Ax)
A3x = np.dot(A, A2x)
# print(np.array(A))
# print(np.array(x))
print('Ax:')
print(Ax)

print('A2x:')
print(A2x)

print('A3x:')
print(A3x)

print('My obversvation does generalize, however one important thing to note is that it
```

```
HIGHLIGHTED QUESTION - Now do the same for x=[0,0,1,0,0]T. Does your observation gene
ralize?
Ax:
[[0]
 [0]
 [0]
 [1]
 [1]]
A2x:
[[0]
 [0]
 [0]
 [0]
 [1]]
A3x:
[[0]
 [0]
 [0]
 [0]
 [0]]
My obversvation does generalize, however one important thing to note is that it will
now represent the number of length 1, 2, 3 or 3 paths for Ax, A^2x, or A^3x respectiv
ely from node 2 to any other node i. So, we would have to be mindful of the node mark
ed as 1 in the column vector x as that will be node i.
```

In [ ]:

In [ ]:

In [131…

```python
import networkx as nx
import sys
sys.path.append('../d3networkx/')
import d3networkx as d3nx
from d3graph import D3Graph, D3DiGraph
from numpy import *
from time import time
import asyncio

def square_grid(n,d3,G,x0=100,y0=100,w=50):
    if G is None:
        G = D3Graph()
    # find the dimensions for the grid that are as close as possible
    num_rows = int(floor(sqrt(n)))
    while n % num_rows != 0:
        num_rows += 1
    num_cols = int(n/num_rows)

    # Add all the nodes
    G.add_nodes_from(range(n))

    # Add the edges and position the nodes
    for i in range(num_rows):
        for j in range(num_cols):
            n = num_cols*i + j
            d3.position_node(n,x0+i*w,y0+j*w)
            if i < num_rows-1:
                G.add_edge(n,n+num_cols) # add edge down
            if j < num_cols-1:
                G.add_edge(n,n+1) # add edge right
```

```
async def propagate(G,d3,x,steps,slp=0.5,keep_highlights=False,update_at_end=False):
    interactive = d3.interactive
    d3.set_interactive(False)
    A = nx.adjacency_matrix(G).todense().T  # adjacency matrix
    d3.highlight_nodes_by_index(list(where(x>0)[0]))
    d3.update()
    await asyncio.sleep(slp)
    cum_highlighted = sign(x)
    for i in range(steps): # the brains
        x = sign(dot(A,x)) # the brains
        cum_highlighted = sign(cum_highlighted+x)
        if not update_at_end:
            if not keep_highlights:
                d3.clear_highlights()
            d3.highlight_nodes_by_index(list(where(x>0)[0]))
            d3.update()
            await asyncio.sleep(slp)
    if update_at_end:
        if not keep_highlights:
            d3.clear_highlights()
            d3.highlight_nodes_by_index(list(where(x>0)[0]))
        else:
            d3.highlight_nodes_by_index(list(where(cum_highlighted>0)[0]))
        d3.update()
    d3.set_interactive(interactive)
    if keep_highlights:
        return cum_highlighted
    else:
        return x
```

This next line starts up the visualizer. It will start some background code that sends data to the visualizer and then it will open a new browser window where the visualizer will live. Once you have the visualizer running, you can leave it running for the entire session, so don't re-run this block. If you close the `visualizer.html` (or hit refresh), you will need to reestablish this connection. In this case, you should click the refresh button in the Jupyter notebook (not for the webpage) to restart the kernel (which will clear your variables and Python environment).

In [132…

```
# d3 = await d3nx.create_d3nx_visualizer()
d3 = await d3nx.create_d3nx_visualizer(canvas_size=(1200,1000))
```

```
websocket server started...visualizer connected...networkx connected...
```

Launching the Visualizer Manually

If the visualizer does not launch automatically, then you'll need to open it manually. After running the line above, use the following line to determine the communication port that the visualizer is using:

In [133…

```
d3.port
```

Out[133]:

```
5124
```

The port is a 4-digit number. Go to the file *visualizer.html* in the d3networkx folder. Double click on it to open it (do not open it in JupyterLabs). In the url, add the following (without the quotes) to the end of the url: "&port=1234" and replace 1234 with the 4-digit port above. A different port is selected each time you run the `d3 = await d3nx.create_d3nx_visualizer()` command. So as long as you don't run that command again (or restart the kernel, or close JupyterLab, that port should still continue to work)

Grid Network

In [134…

```
d3.clear()
d3.set_interactive(False)
G = D3Graph()
d3.set_graph(G)
square_grid(144,d3,G,x0=75,y0=70)
d3.update()
```

In [135…

```
# x = zeros((G.number_of_nodes(),1))
# x[0] = 1
# A = nx.adjacency_matrix(G).todense().T  # adjacency matrix
# print(list(A))
# print(x)
# for i in range(3):

#     x = sign(dot(A,x)) # the brains
#     print(x)
```

In [136…

```
x = zeros((G.number_of_nodes(),1))
x[0] = 1
chk = await propagate(G,d3,x,13,slp=1);
print(chk)
```

```
C:\Users\Humza\AppData\Local\Temp\ipykernel_7228\3358298058.py:35: FutureWarning: adj
acency_matrix will return a scipy.sparse array instead of a matrix in Networkx 3.0.
  A = nx.adjacency_matrix(G).todense().T  # adjacency matrix
```

```
[[0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]
```

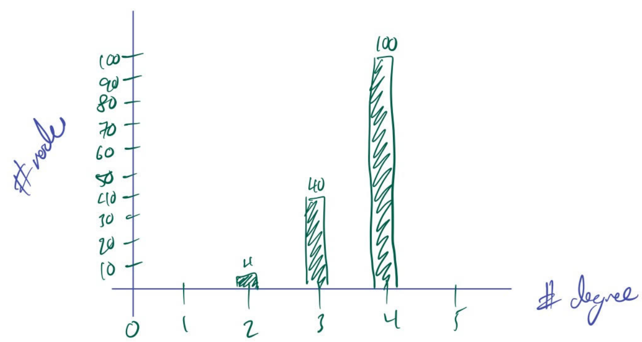
```
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
```

```
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```

In [17]:
```
print('HIGHLIGHTED QUESTION - By hand, looking at the network, write down the degrees
print()
print('4 nodes have degree 2.\n40 nodes have degree 3.\n100 Nodes have degree 4.')
```

```
HIGHLIGHTED QUESTION - By hand, looking at the network, write down the degrees of the
nodes (you can aggregate your answer, so write down how many nodes have degree k). Pi
ck a way to visualize this aggregated degree data (a graph of some sort) - just draw
something by hand.

4 nodes have degree 2.
40 nodes have degree 3.
100 Nodes have degree 4.
```



In [137…

```
print('HIGHLIGHTED QUESTION - Briefly describe why you see the pattern that you do and
print()
print('The pattern alternates in highlighting the diagonals starting from the bottom-l
```

HIGHLIGHTED QUESTION - Briefly describe why you see the pattern that you do and indicate why the process stops where it does. How could you make the propagation go further?

The pattern alternates in highlighting the diagonals starting from the bottom-left and builds upon the previously highlighted diagonals. It does this because x = sign(dot(A,x)) will createa column vector of nodes that need to be highlighted. When we start with node 0 (bottom-left) as the only 1 in the column vector, the dot product will give us a list of all the nodes adjacent to node 0. Then, when we reassign this new column vector to x and again take the dot product of A and x, we receive a column vector of all the neighbors of all nodes in column vector x. This will then result in highlighting nodes in alternating manners such that we will see "even" and "odd" diagonals being highlighted when visualizing. This process stops where it does because we provide a maximum number of 10 steps. To make the propogation go further we simply have to chose our maximum steps to be greater than 10.

In [ ]:

Directed Network

In [138…

```
d3.clear()
G = D3DiGraph(nx.read_weighted_edgelist('lab2.edgelist',create_using=nx.DiGraph))
d3.set_graph(G)
d3.update()
```

In [139…

```
print('HIGHLIGHTED QUESTION - Now, in the next code block, add code to propagate this
```

```
HIGHLIGHTED QUESTION - Now, in the next code block, add code to propagate this network starting at node 0 for 10 steps.
```

In [140…

```
# code to propagate
x = zeros((G.number_of_nodes(),1))
x[0] = 1
await propagate(G, d3, x, 10, slp=1)
```

```
C:\Users\Humza\AppData\Local\Temp\ipykernel_7228\3358298058.py:35: FutureWarning: adjacency_matrix will return a scipy.sparse array instead of a matrix in Networkx 3.0.
  A = nx.adjacency_matrix(G).todense().T  # adjacency matrix
```

Out[140]:

```
matrix([[0.],
        [0.],
        [0.],
        [0.],
        [0.]])
```

In [141…

```
A = nx.adjacency_matrix(G).todense().T  # adjacency matrix
# print(A)
print(A**5)
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

```
C:\Users\Humza\AppData\Local\Temp\ipykernel_7228\3501486504.py:1: FutureWarning: adjacency_matrix will return a scipy.sparse array instead of a matrix in Networkx 3.0.
  A = nx.adjacency_matrix(G).todense().T  # adjacency matrix
```

In [142…

```
print('HIGHLIGHTED QUESTION - What is the end result? What is A^10 (A is the adjacency
print()
print('The end result returned is a column vector filled with 0s. A^10, as seen above
```

```
HIGHLIGHTED QUESTION - What is the end result? What is A^10 (A is the adjacency matri
x)? Describe why this is the case from the point of view of network paths and also fr
om linear algebra (i.e., what special name do we give such matrices?).

The end result returned is a column vector filled with 0s. A^10, as seen above is a m
atrix filled with 0s. From the point of view of network paths, we find that in this d
irected network that when trying to find paths 10 steps away from node 0 we end at no
de 4, but cannot find any paths beyond a length of 4 starting from node 0 since there
is no out-neighbor for node 4. From a linear algebra perspective we get the null matr
ix (a matrix filled with 0s) which describes that no paths of length 10 starting from
node 0 exist in the network.
```

In [143…

```
# code to find the out-component of node 1
x = zeros((G.number_of_nodes(),1))
x[1] = 1
await propagate(G, d3, x, 20, update_at_end=True, keep_highlights=True)
```

```
C:\Users\Humza\AppData\Local\Temp\ipykernel_7228\3358298058.py:35: FutureWarning: adj
acency_matrix will return a scipy.sparse array instead of a matrix in Networkx 3.0.
  A = nx.adjacency_matrix(G).todense().T  # adjacency matrix
```

Out[143]:

```
matrix([[0.],
        [1.],
        [1.],
        [1.],
        [1.]])
```

In [144…

```
print('HIGHLIGHTED QUESTION - Run this code - is the out-component what you predicted?
print()
print('I predicted the out-component to consist of nodes 2, 3, and 4, but it is actual
print()
print('HIGHLIGHTED QUESTION - Without changing the code within the propagate(...) func
print()
print('In-components are nodes which have a directed path to a given node, including t
print()
print('HIGHLIGHTED QUESTION - What is the size of the largest strongly connected compo
print()
print('The size of the largest strongly connected component is 1')
```

```
HIGHLIGHTED QUESTION - Run this code - is the out-component what you predicted?

I predicted the out-component to consist of nodes 2, 3, and 4, but it is actually nod
es 1, 2, 3, and 4 which makes sense since node 1 can reach itself.

HIGHLIGHTED QUESTION - Without changing the code within the propagate(...) function,
how could we use it to find in-components instead of out-components?

In-components are nodes which have a directed path to a given node, including the giv
en node itself. One way to find the in-component of a given node would be to execute
propogate for all other nodes j and see if the given node i is part of the out-compon
ent, if it is then a path exists; so, we can add node j to the list of in-component f
or node i. By default we add the given node i to the in-component list.

HIGHLIGHTED QUESTION - What is the size of the largest strongly connected component?

The size of the largest strongly connected component is 1
```

E. coli Protein Network

In [145…

```
d3.clear()
G = D3DiGraph(nx.read_weighted_edgelist('ecoli.edgelist',create_using=nx.DiGraph))
d3.set_interactive(False)
d3.set_graph(G)
d3.set_interactive(True)
d3.update()
print('Ecoli has %i nodes.' % G.number_of_nodes())
```

```
Ecoli has 418 nodes.
```

In [146…

```
print('HIGHLIGHTED QUESTION - In the next two code blocks, find the out-components of
```

```
HIGHLIGHTED QUESTION - In the next two code blocks, find the out-components of nodes
with index 2 and 16. Have your program print out the size of the component
```

In [147…

```
# code to find the out-component of node 2
x = zeros((G.number_of_nodes(),1))
x[2] = 1
col_vec = await propagate(G, d3, x, 1, update_at_end=True, keep_highlights=True)
out_component = list(where(col_vec>0)[0])
out_component_nodes = [G.node_by_index(i) for i in out_component]
print(f'Out-component of node with index 2: {out_component_nodes} which has a size of
```

```
C:\Users\Humza\AppData\Local\Temp\ipykernel_7228\3358298058.py:35: FutureWarning: adj
acency_matrix will return a scipy.sparse array instead of a matrix in Networkx 3.0.
  A = nx.adjacency_matrix(G).todense().T  # adjacency matrix
Out-component of node with index 2: ['6', '11', '14'] which has a size of 3
```

In [148…

```
G.node_by_index(16)
out_component_nodes = [G.node_by_index(i) for i in out_component]
print(G.node_by_index(2), G.node_by_index(3), G.node_by_index(4))
print(out_component)
```

```
6 11 14
[2, 3, 4]
```

In [149…

```
d3.clear_highlights()
d3.update()
```

In [150…

```
# code to find the out-component of node 16
x = zeros((G.number_of_nodes(),1))
x[16] = 1
col_vec = await propagate(G, d3, x, 418, update_at_end=True, keep_highlights=True)
out_component = list(where(col_vec>0)[0])
out_component_nodes = [G.node_by_index(i) for i in out_component]
print(f'Out-component of node with index 16: {out_component_nodes} which has a size of
```

```
C:\Users\Humza\AppData\Local\Temp\ipykernel_7228\3358298058.py:35: FutureWarning: adj
acency_matrix will return a scipy.sparse array instead of a matrix in Networkx 3.0.
  A = nx.adjacency_matrix(G).todense().T  # adjacency matrix
Out-component of node with index 16: ['18', '17', '24', '1', '49', '73', '76', '85',
'117', '144', '152', '153', '165', '172', '177', '202', '217', '242', '285', '351',
'357', '50'] which has a size of 22
```

In [151…

```
print('HIGHLIGHTED QUESTION - What is the minimum value of steps that guarantees that
print()
```

```
print('Through trial and error, I found that for out-component of node index 2, the mi
```

```
HIGHLIGHTED QUESTION - What is the minimum value of steps that guarantees that you wi
ll find the entire out-component?

Through trial and error, I found that for out-component of node index 2, the minimum
value of steps is 1. For the entire out-component of node index 16 the minimum value
of steps is 2.
```

In [ ]:

```

```

In [152…

```
print('HIGHLIGHTED QUESTION - find and display the diameter of this network')
```

```
HIGHLIGHTED QUESTION - find and display the diameter of this network
```

In [158…

```
#print(nx.diameter(G)) # does not work!

def diameter2(G):
    spaths = dict(nx.all_pairs_shortest_path(G))

    path = []
    diameter = 0

    # iterate nested dictionary
    for k in spaths:
        for v in spaths[k]:
            curr_path = spaths[k][v] # get current path
            length = len(curr_path) - 1 # get current path length
            if length > diameter: # compare to current diameter
                diameter = length # update diameter
                path = curr_path # update longest shortest path
    return diameter, path

# use the new diameter function here
```

In [159…

```
d, p = diameter2(G)
print(f'Diameter: {d}')
print(f'Path: {p}')
d3.highlight_nodes(p)
edges = []
for i in range(len(p)-1):
    edges.append((p[i], p[i+1]))

d3.highlight_edges(edges)
```

```
Diameter: 4
Path: ['190', '292', '136', '137', '133']
```

Section 6.12: Flows & Cut Sets

In [160…
```
from networkx import bipartite
```

In [161…
```
# creates a directed complete graph
def worst_graph(n):
    # to do
    G = nx.DiGraph()
    G.add_nodes_from(range(n))
    G.add_edges_from([(i, j) for i in range(n) for j in range(n) if i != j], capacity=
    return G

# creates a directed path graph
def best_graph(n):
    G = nx.DiGraph()
    nx.add_path(G, list(range(n)), capacity=1)

    return G

# create graph


worst = worst_graph(1000)
best = best_graph(1000)

# G_best = D3DiGraph(best)
# G_worst = D3DiGraph(worst)
# d3.set_interactive(False)
# d3.set_graph(G_worst)
# d3.set_interactive(False)
# d3.update()

# print(worst)
```

In [162…
```
print('HIGHLIGHTED QUESTION - Describe these graphs in words')
print()
print('The worst graph is a directed complete graph without weights and a capacity of
print()


print('HIGHLIGHTED QUESTION - Choose n to be somewhat large (1,000) and observe the co
```

```
# perform min cut for worst
start_time = time()
nx.minimum_cut(worst, 0, 999)
print('WORST - min cut took %1.2f seconds' % (time() - start_time))

# perform min cut for best
start_time = time()
nx.minimum_cut(best, 0, 999)
print('BEST - min cut took %1.2f seconds' % (time() - start_time))
```

```
HIGHLIGHTED QUESTION - Describe these graphs in words

The worst graph is a directed complete graph without weights and a capacity of 1 per
edge, making it a weakly connected directed unweighted graph. The best graph is a dir
ected path graph without weights and a capacity of 1 per edge, making it a weakly con
nected unweighted graph.

HIGHLIGHTED QUESTION - Choose n to be somewhat large (1,000) and observe the computat
ion time required on these two types of graphs
WORST - min cut took 4.91 seconds
BEST - min cut took 0.02 seconds
```

In [163…

```
print('HIGHLIGHTED QUESTION - First, reason why this graph of activities should be an
print()
print('A DAG has no cycles by definition so there will always be a well-defined sequer
```

```
HIGHLIGHTED QUESTION - First, reason why this graph of activities should be an acycli
c directed graph. Explain why cycles in a project activity network would be a bad way
to organize it.

A DAG has no cycles by definition so there will always be a well-defined sequence fro
m a starting node to a final node. If we had a cycle present in a project activity ne
twork then it would create confusion in terms of the ordering of the tasks. Cycles ca
n also cause infinite loops, so a project may never end. Overall, DAGs help us avoid
circular dependencies.
```

In [164…

```
G = nx.read_gml('pert.gml', 'name')


print('HIGHLIGHTED QUESTION - Using the Bellman-Ford shortest path algorithm, find the
print()

path = nx.bellman_ford_path(G, 'Lead time', 'Leave site', weight='weight')
length = nx.bellman_ford_path_length(G, 'Lead time', 'Leave site', weight='weight')
print(path)
print(length)
```

```
HIGHLIGHTED QUESTION - Using the Bellman-Ford shortest path algorithm, find the lengt
h and activity sequence of the critical path in this activity network

['Lead time', 'Obtain valves', 'Fit valves', 'Finish valve chambers', 'Leave site']
48.0
```

In [165…

```
print('HIGHLIGHTED QUESTION - calculate the minimum cut set and the cut set weight fro
print()
cut_value, partition = nx.minimum_cut(G, 'Lead time', 'Leave site', capacity='weight')
reachable, non_reachable = partition

cutset = set()

for u, nbrs in ((n, G[n]) for n in reachable):
```

```
    cutset.update((u, v) for v in nbrs if v in non_reachable)

weight = 0
for i,j in cutset:
    weight += G[i][j]['weight']

print(f'Minimum Cutset: {sorted(cutset)}')
print(f'Cutset weight: {weight}')
```

```
HIGHLIGHTED QUESTION - calculate the minimum cut set and the cut set weight from “Lea
d time” to “Leavesite”

Minimum Cutset: [('Clean up', 'Leave site'), ('Finish valve chambers', 'Leave site')]
Cutset weight: 11.0
```

In [166…

```
print('HIGHLIGHTED QUESTION - Can you think of a potential use-case for analyzing the
print()
print('The minimum cut of an activity network can help us determine which activities a
```

```
HIGHLIGHTED QUESTION - Can you think of a potential use-case for analyzing the minimu
m cut of an activity network?

The minimum cut of an activity network can help us determine which activities are ess
ential and then prioritize resources to make sure the project remains on a good timel
ine
```

In [ ]:

```
```