Machine Learning on Graphs - Final 2

Mohammad Bahrami - 9724133

January 12, 2022

---

1 Question 2

1.1

The main Graph of the Question is show in figure 1.

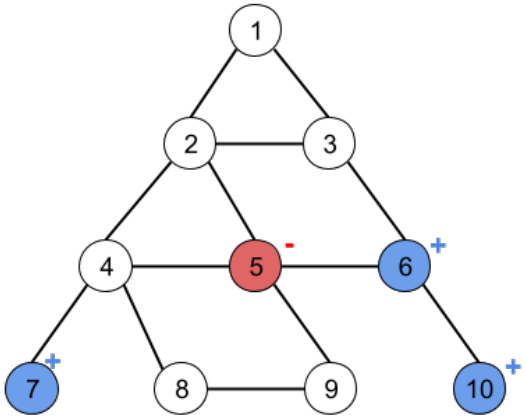


Figure 1: Original Graph

As the Questions Asks, we initialize all blue(+) nodes with 1.0 and all red (−) nodes with 0.0 And all other nodes with 0.5. This is depicted in figure 2.

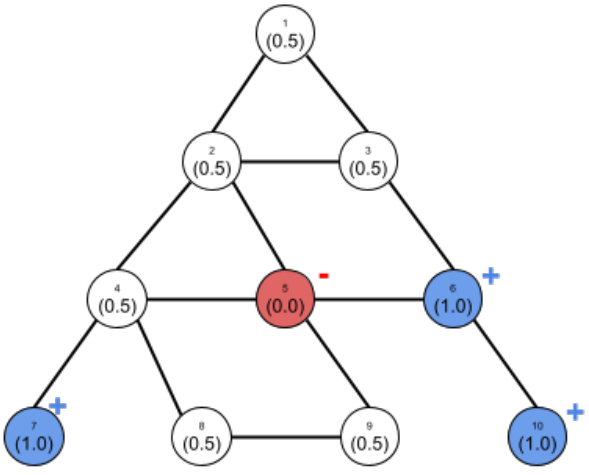


Figure 2: Relational Classification initialization State (Iteration 0)

we perform two iterations of relational classification. the result of each iteration is shown in below figures.

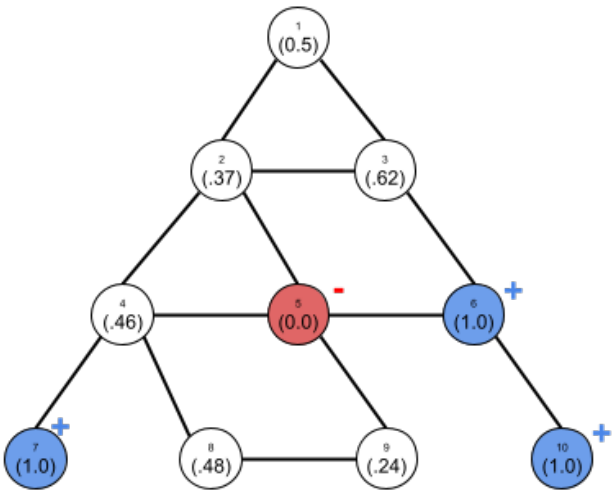


Figure 3: Iteration 1

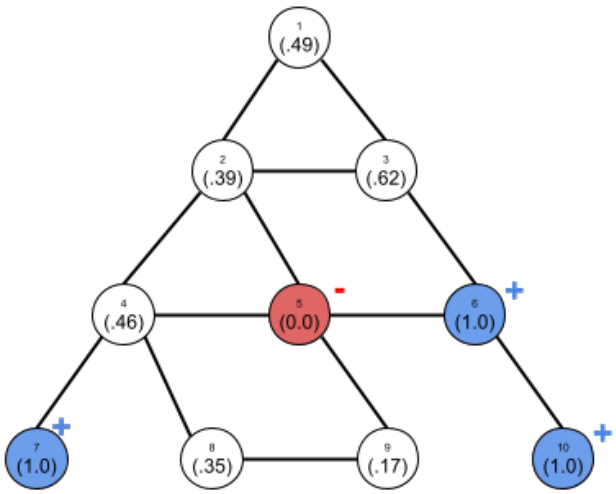


Figure 4: Iteration 2

we can see that

$$P(Y_3 = +) = 0.62$$
$$P(Y_4 = +) = 0.46$$
$$P(Y_8 = +) = 0.35$$

1.2

Now we put every node v that $P(Y_v = +) > 0.5$ to be in the blue $(+)$ class, and all other nodes to be in red $(-)$ class. this is shown in figure 5.

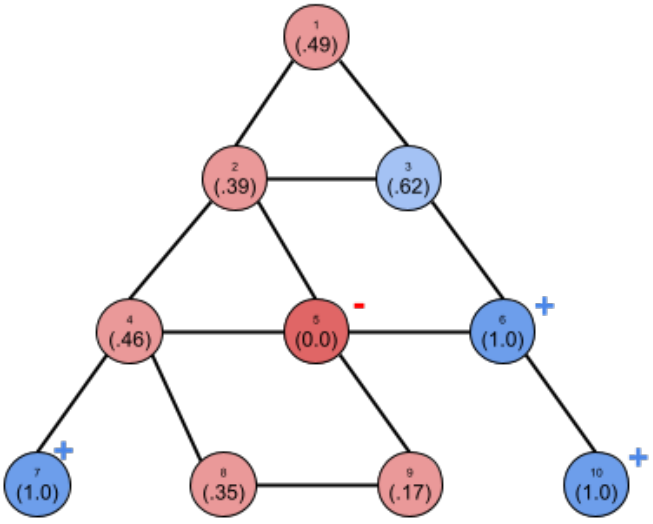


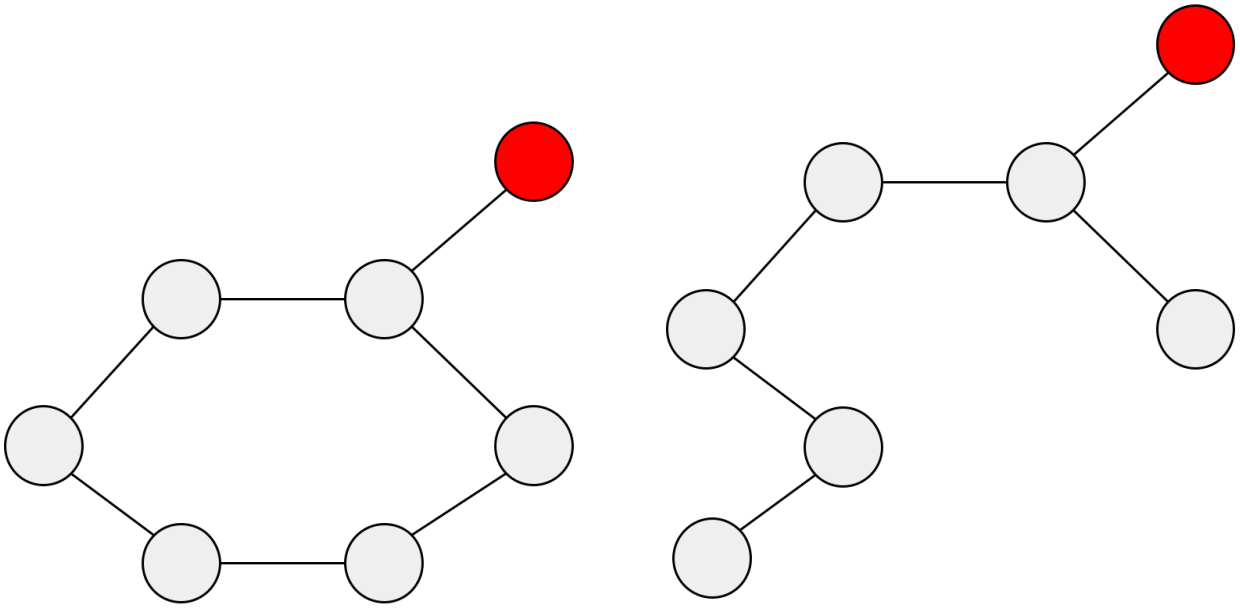
Figure 5: Final result of Relational Classification after two iterations

We can see that nodes $1, 2, 4, 5, 8, 9$ are classified as red $(-)$.

2 Question 4

2.1

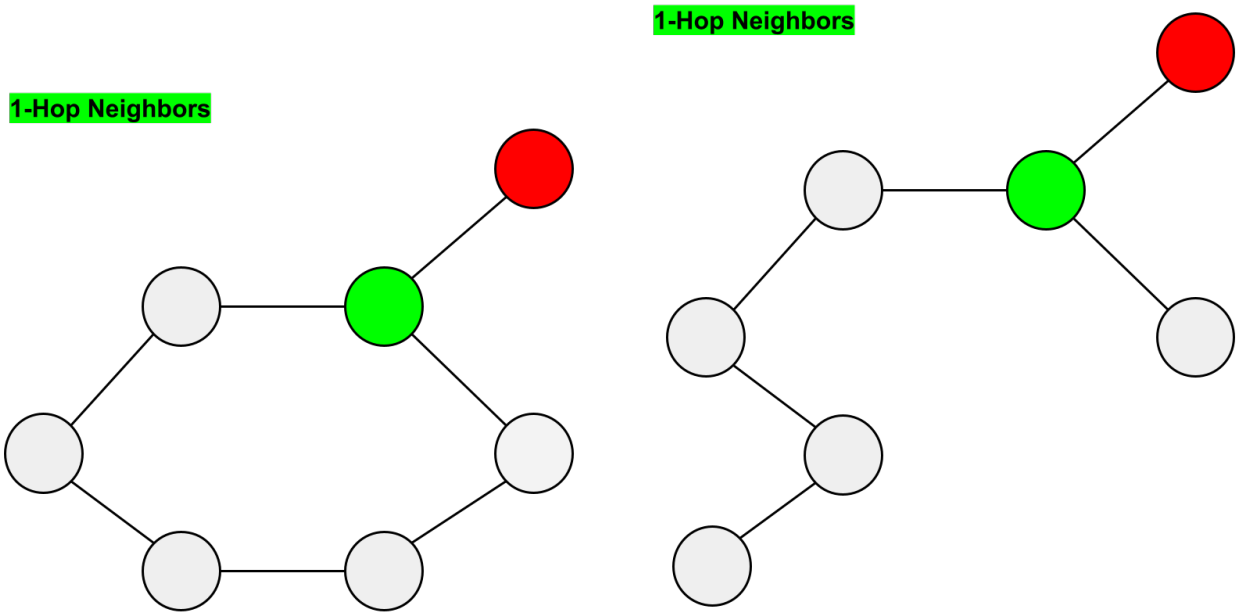
We try to highlight K-hop neighbors in the following figures and we try to find the first occurrence of a different neighborhood. The Original Graphs are shown in figure 6.



(a) Graph 1 (b) Graph 2

Figure 6: Original Graphs

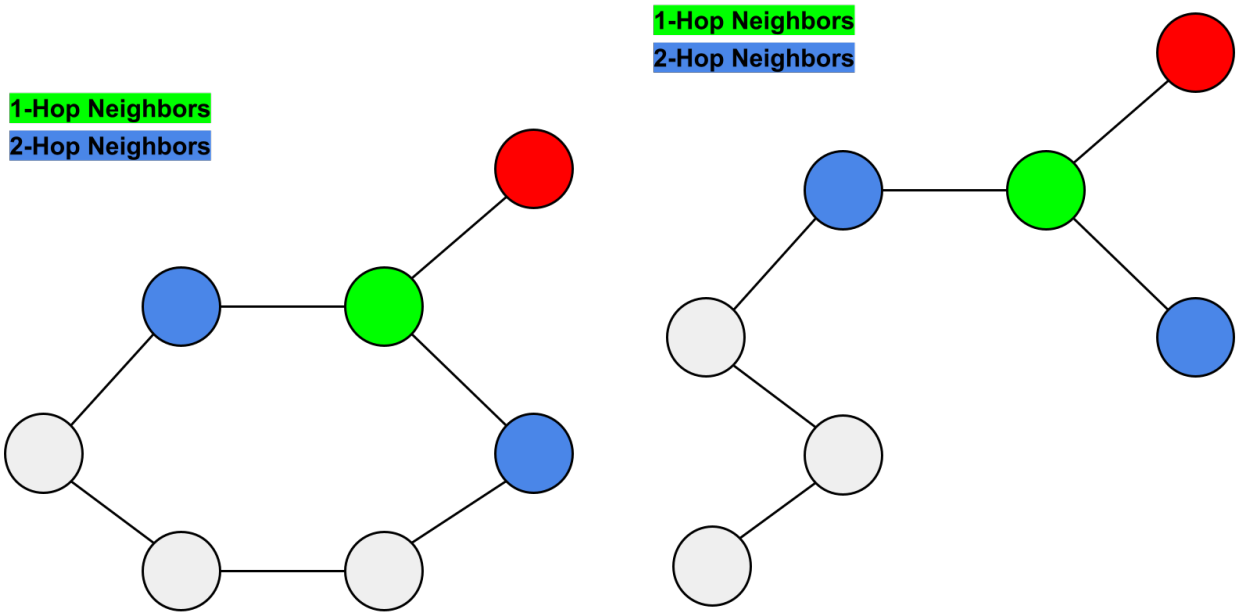
First, We highlight the 1-Hop Neighbors in both graphs, as shown in figure 7.



(a) Graph 1 (b) Graph 2

Figure 7: Graphs with 1-hop neighbors highlighted

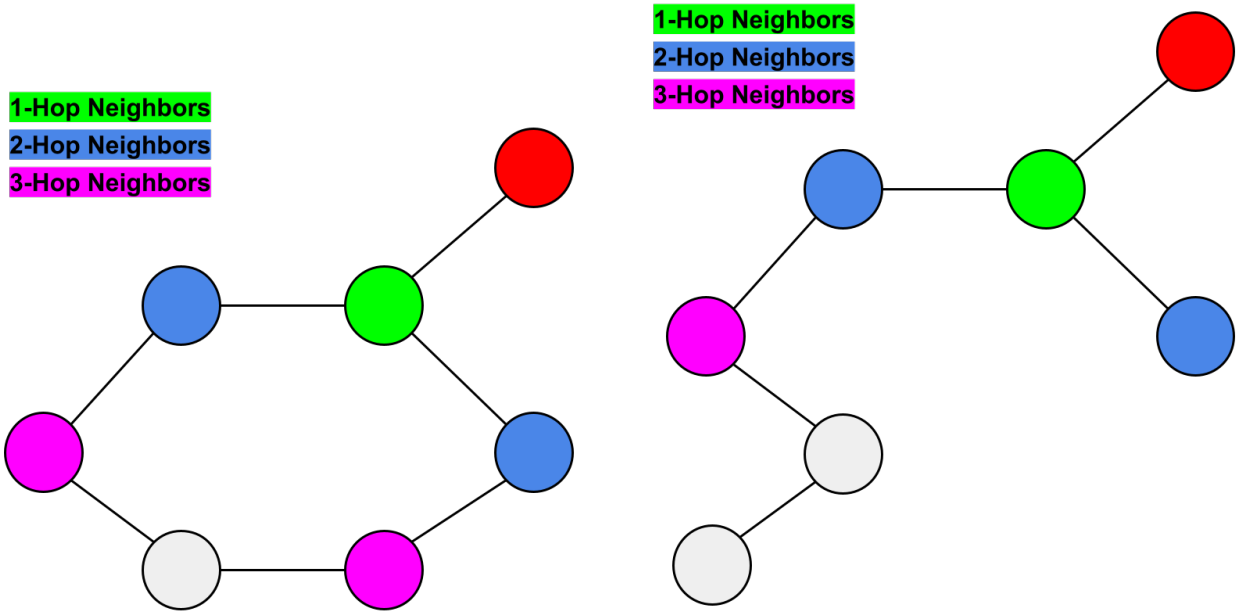
We can see that the structure of the 1-hop neighborhood with respect to the red node is exactly the same. Thus, we need to continue highlighting.



(a) Graph 1 (b) Graph 2

Figure 8: Graphs with 2-hop neighborhood highlighted

As depicted in figure 8, 2-hop neighborhood is the same too. So we move on to 3-hop neighborhood.



(a) Graph 1 (b) Graph 2

Figure 9: Graphs with 3-hop neighborhood highlighted

As shown in figure 9, We can see that in the 3-hop neighborhood with respect to the red node, in graph (a) there are two pink nodes, where as in graph (b) there is only one pink node. This is the first occurrence of any difference in the k-hop neighborhood with respect to the red node. so the minimum K for the embedding of the red node to be different in graphs (a) and (b) is 3.

I have Also written a short script to check the validity of this answer. The code can be found at `./Final2-MohammadBahrami-9724133.ipynb> Question 4.1`

2.2

2.2.1

To find the Random Walk Transition Matrix, first we write the adjacency matrix of the graph.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Now we normalize each column.

$$M = \begin{bmatrix} 0.0 & 0.5 & 0.0 & 0.33 \\ 0.5 & 0.0 & 0.0 & 0.33 \\ 0.0 & 0.0 & 0.0 & 0.33 \\ 0.5 & 0.5 & 1.0 & 0.0 \end{bmatrix}$$

$$M \cdot r = \begin{bmatrix} 0.0 & 0.5 & 0.0 & 0.33 \\ 0.5 & 0.0 & 0.0 & 0.33 \\ 0.0 & 0.0 & 0.0 & 0.33 \\ 0.5 & 0.5 & 1.0 & 0.0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

2.2.2

$$M \cdot r = \lambda \cdot r \xrightarrow{\lambda=1} M \cdot r = r \implies r = \begin{bmatrix} 0.47 \\ 0.47 \\ 0.24 \\ 0.71 \end{bmatrix}$$

2.3

We need to obtain a stochastic matrix for M. For each column i of M, $M_{i,j}$ will be $\frac{A_{i,j}}{D_{j,j}}$ for all js. Where D is a diagonal matrix where $D_{i,i}$ is degree of each node. If we want to vectorize this, we will get

$$M = A \cdot D^{-1}$$

2.4

This is exactly like the previous question, except that now every node can get messages from itself too. this means that each node has a connection to itself but with a weight of $\frac{1}{2}$. we can simulate this by adding $\frac{1}{2}I$ to the result of the previous question, which will result into

$$M = \frac{1}{2}A \cdot D^{-1} + \frac{1}{2}I$$

2.5

TODO

2.6

1. **Message:** Each node will send a 1 to all its neighbors if it has been seen and a 0 otherwise.
2. **Aggregation:** Each node will perform logical OR on all the received messages.
3. **Update:** Each node will update its value to the logical OR of its previous value and the result of the aggregation.

If we Consider a vector named BFS that the nodes that have been seen have 1 in their respective index in the BFS vector and the nodes that have not been seen have 0, we initialize the BFS vector with the index of source node 1 and 0 for all other nodes. then for each layer, we can define the BFS of the next layer as:

$$BFS_u^{(l+1)} = BFS_u^{(l)} \vee (\vee_{v \in N(u)} BFS_v^{(l)})$$

The code for this solution is available at `./Final2-MohammadBahrami-9724133.ipynb> Question 4.6`

Also, as a better idea, we can add the result of the previous layer as the following:

$$BFS_u^{(l+1)} = BFS_u^{(l)} + (BFS_u^{(l)} \vee (\vee_{v \in N(u)} BFS_v^{(l)}))$$

The stop condition will be when there is no more 0s in the BFS vector. After that, we can arg sort the BFS vector with a descending order to obtain the order of which the nodes were visited.

3

I will Append an Export of my .ipynb file from the next page on for convenience.

ipynb-pdf

January 12, 2022

[ ]:
```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
```

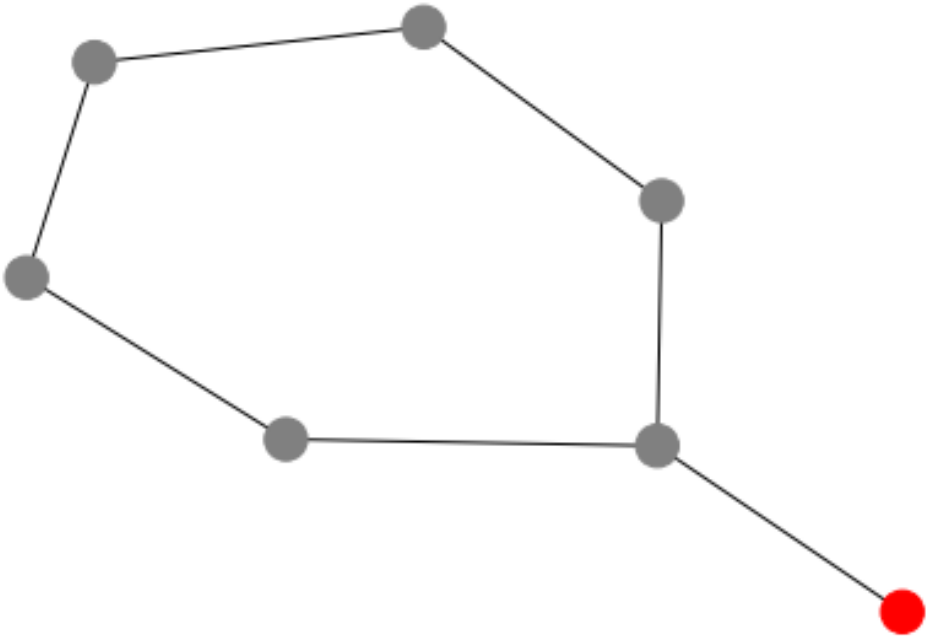
0.1 Question 4.1

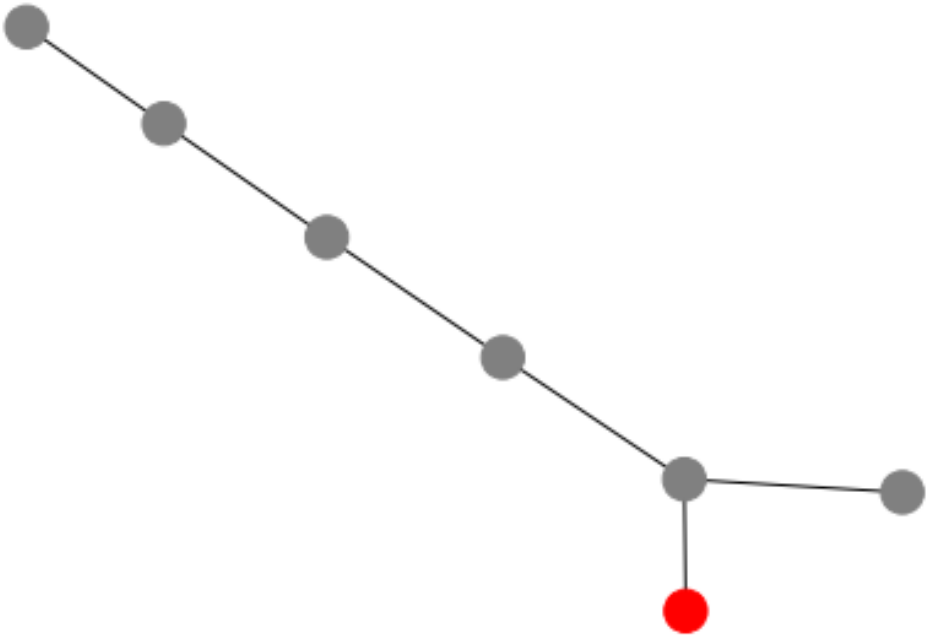
[ ]:
```
a = nx.Graph()
a.add_nodes_from(list(range(1,8)))
a.add_edges_from([
    (1, 2),
    (2, 3),
    (2, 4),
    (3, 5),
    (5, 6),
    (6, 7),
    (7, 4)
])

b = nx.Graph()
b.add_nodes_from(list(range(1,8)))
b.add_edges_from([
    (1, 2),
    (2, 3),
    (2, 4),
    (3, 5),
    (5, 6),
    (6, 7)
])

a_adj = nx.to_numpy_array(a)
b_adj = nx.to_numpy_array(b)

a_x = np.ones((7, ))
b_x = np.ones((7, ))

watch_node_idx = 0
iter=0
```

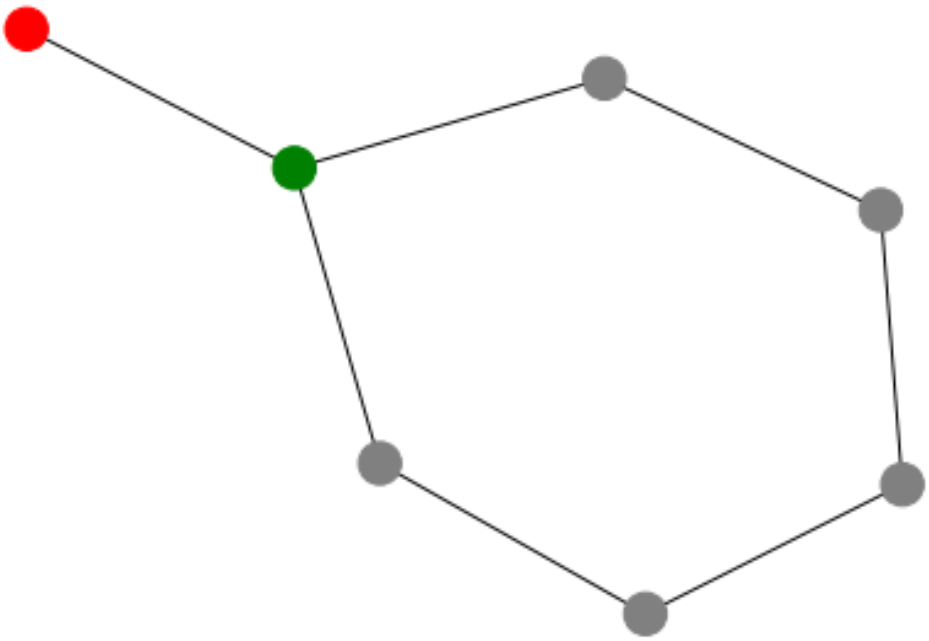
```
while(a_x[watch_node_idx] == b_x[watch_node_idx]):
    a_x = a_x + np.sum(a_adj * a_x, axis=1)
    b_x = b_x + np.sum(b_adj * b_x, axis=1)
    iter += 1
print(f'First difference between red node of graph a, b happend in iteration:␣
 ↪{iter}')
```

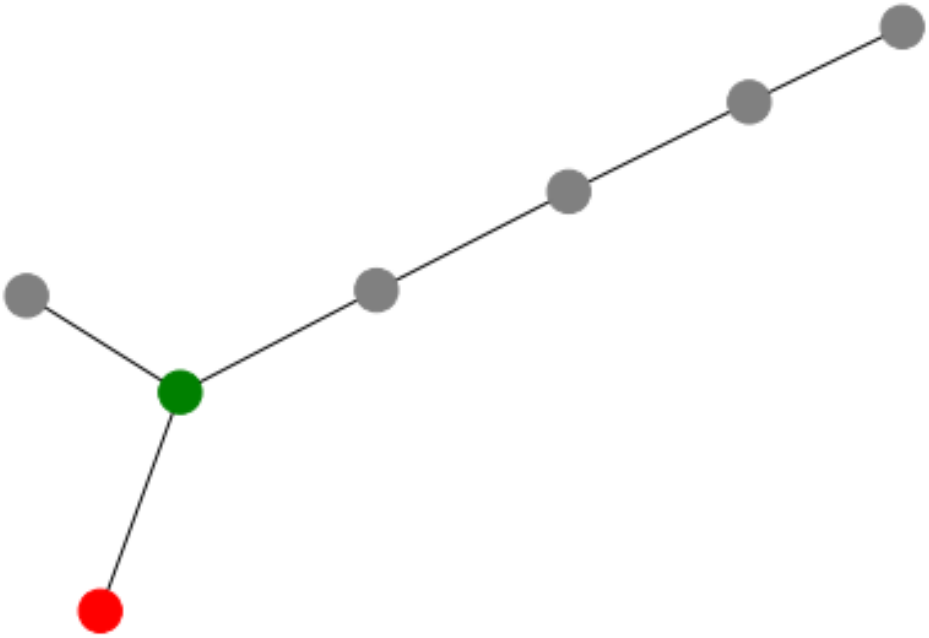
```
First difference between red node of graph a, b happend in iteration: 3
```

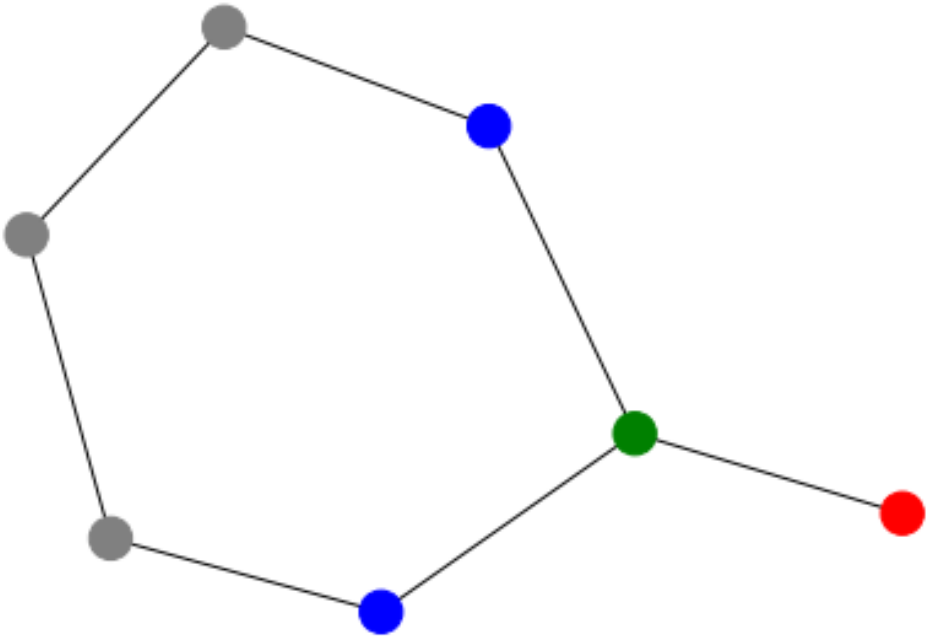
[ ]:
```
colors = ['red', 'grey','grey','grey','grey','grey','grey']
nx.draw(a, node_color=colors)
plt.show()
plt.clf()
nx.draw(b, node_color=colors)
plt.show()
plt.clf()

colors = ['red', 'green','grey','grey','grey','grey','grey']
nx.draw(a, node_color=colors)
plt.show()
plt.clf()
nx.draw(b, node_color=colors)
plt.show()
plt.clf()

colors = ['red', 'green','blue','blue','grey','grey','grey']
nx.draw(a, node_color=colors)
plt.show()
plt.clf()
nx.draw(b, node_color=colors)
plt.show()
plt.clf()

colors = ['red', 'green','blue','blue','pink','grey','pink']
nx.draw(a, node_color=colors)
plt.show()
plt.clf()
colors = ['red', 'green','blue','blue','pink','grey','grey']
nx.draw(b, node_color=colors)
plt.show()
plt.clf()
```

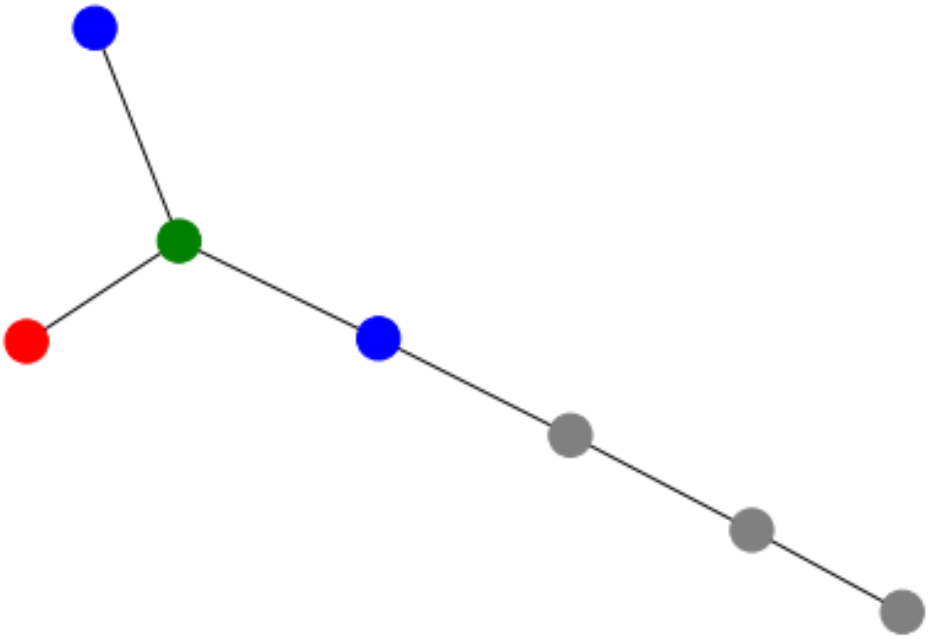


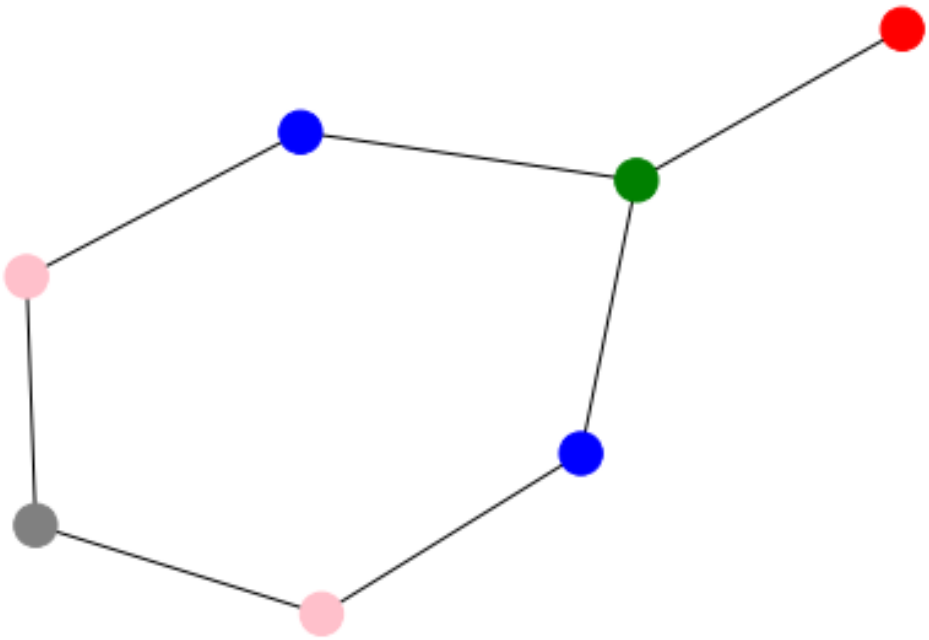


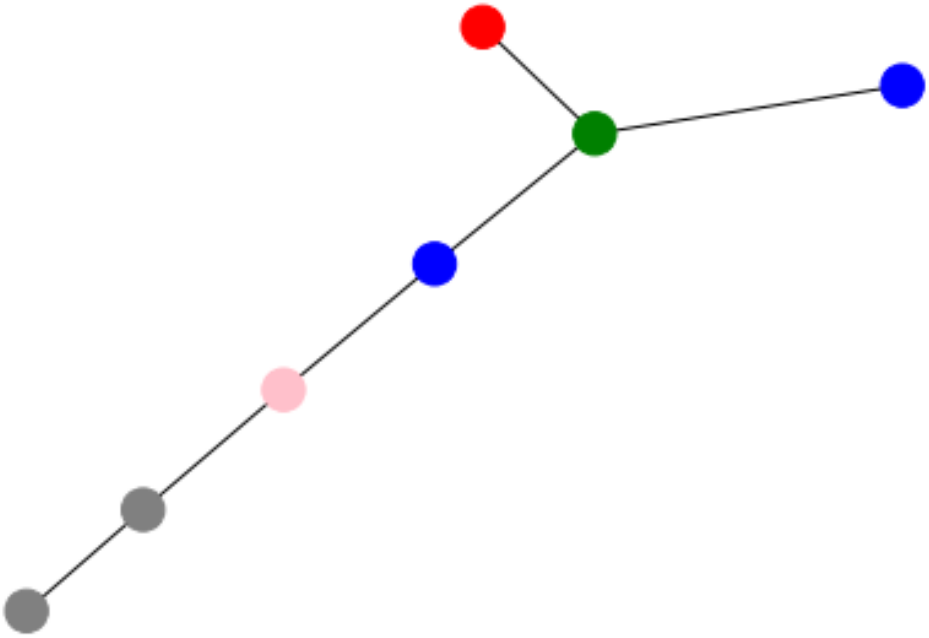












```
<Figure size 432x288 with 0 Axes>
```

0.2 Question 4.2

[ ]:
```
A = np.array([
    [0, 1, 0, 1],
    [1, 0, 0, 1],
    [0, 0, 0, 1],
    [1, 1, 1, 0]
])
M = A / np.sum(A, axis=0)
r = np.array([0, 0, 1, 0])
M@r
```

[ ]:
```
array([0., 0., 0., 1.])
```

[ ]:
```
evals, evecs = np.linalg.eig(M)
print(np.round(evals, 5))
np.round(evecs[:, np.where(np.round(evals, 5) == 1)[0]], 2)
```

```
[ 1.       0.22871 -0.5     -0.72871]
```

[ ]:
```
array([[0.47],
       [0.47],
       [0.24],
       [0.71]])
```

0.3 Question 4.3 and 4.4

[ ]:
```
D = np.eye(4) * np.sum(A, axis=1)
A @ np.linalg.inv(D)
```

[ ]:
```
array([[0.        , 0.5       , 0.        , 0.33333333],
       [0.5       , 0.        , 0.        , 0.33333333],
       [0.        , 0.        , 0.        , 0.33333333],
       [0.5       , 0.5       , 1.        , 0.        ]])
```

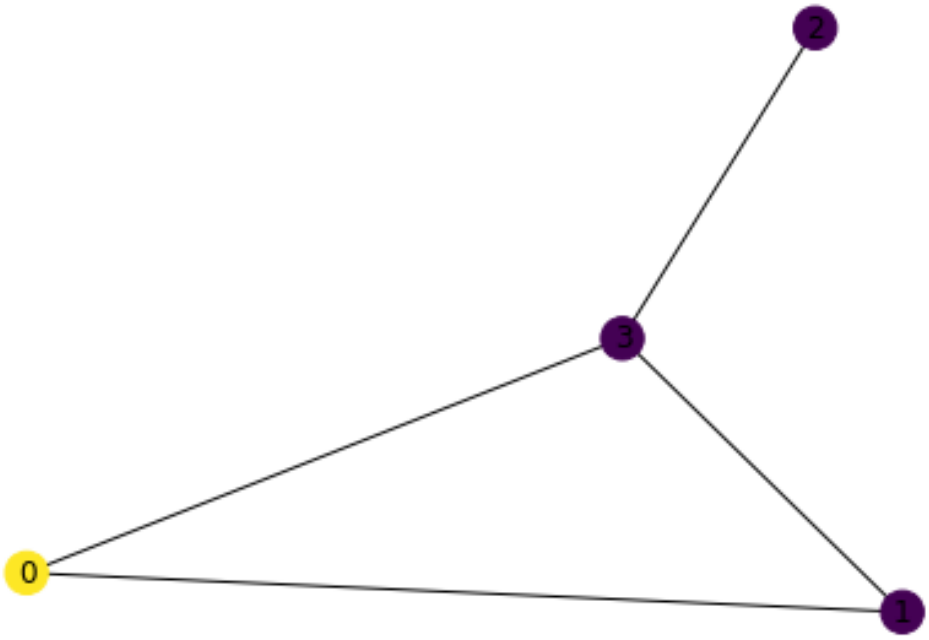
[ ]:
```
1/2*(A @ np.linalg.inv(D)) + 1/2*np.eye(4)
```

[ ]:
```
array([[0.5       , 0.25      , 0.        , 0.16666667],
       [0.25      , 0.5       , 0.        , 0.16666667],
       [0.        , 0.        , 0.5       , 0.16666667],
       [0.25      , 0.25      , 0.5       , 0.5       ]])
```

0.4 Question 4.6

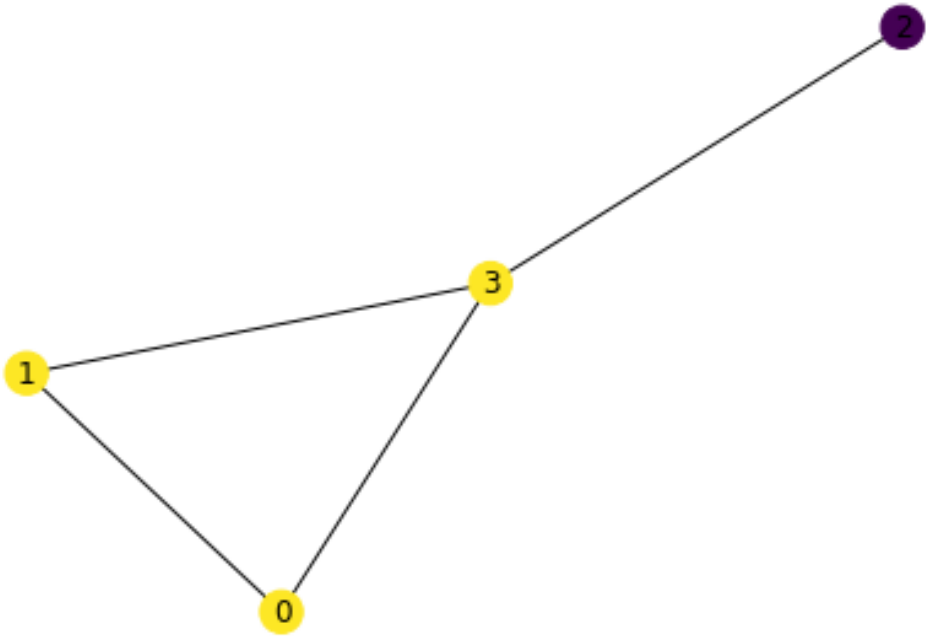
[ ]:
```
BFS = np.zeros((4, )).astype('int')
BFS[0] = 1
BFS, A, nx.draw(nx.from_numpy_array(A),with_labels=True, node_color=BFS)
```

[ ]:
```
(array([1, 0, 0, 0]),
 array([[0, 1, 0, 1],
        [1, 0, 0, 1],
        [0, 0, 0, 1],
        [1, 1, 1, 0]]),
 None)
```



[ ]:
```
#One Step - Run Several times for Several Steps
BFS = np.logical_or(BFS, np.any((A * BFS), axis=1)).astype('int')
BFS, nx.draw(nx.from_numpy_array(A),with_labels=True, node_color=BFS)
```

[ ]:
```
(array([1, 1, 0, 1]), None)
```


2
3
1
0