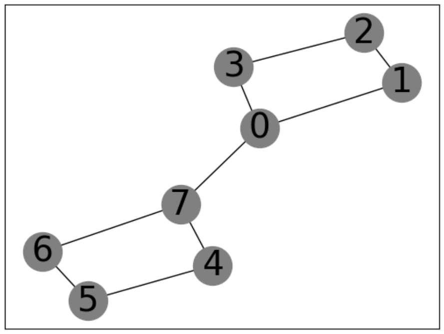
Final Exam (Graph Mining – Spring 2023): Solutions

Full Name:
Student ID:

- The formula and solution process should be presented with the answer.
- All the codes must include detail comments in English.

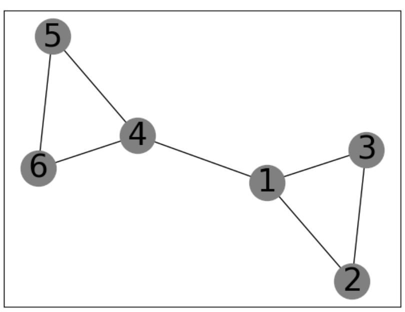
1. Consider an undirected graph G of eight nodes given in the following figure. There are two communities in the graph: A = {0,1,2,3} and B = {4,5,6,7}. Calculate Min-cut and Normalized cut measurements. (10pt)



Min_cut (A,B) = 1

N_cut(A,B) = $\frac{1}{1+4}+\frac{1}{1+4}=\frac{2}{5}=0.4$

2. Consider an undirected graph G of six nodes given in the following figure with two communities: A = {1, 2, 3} and B = {4, 5, 6}. Apply the Equation (1) to calculate the modularity Q of the two communities. (10pt)



$$Q=\frac{1}{2m}\sum_{i,j}\left(A_{ij}-\frac{d_i d_j}{2m}\right)\cdot\delta\left(v_i,v_j\right) \tag{1}$$

$$\delta\left(v_i,v_j\right)=\begin{cases}1 & \text{if } v_i \text{ and } v_j \text{ are in the same community.}\\ 0 & \text{otherwise.}\end{cases}$$

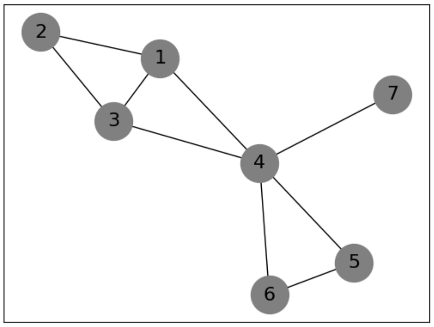
where m is the number of edges, A is the adjacency matrix of G, d_i is the degree of node v_i

```
m=7; anfa = 2 * m
A =[1,2,3]
B= [4,5,6]
Q = 0
adj = nx.adjacency_matrix(G)
adj = adj.todense()
print(adj)
for e in G.edges:
    node1 = e[0]
```

```
    node2 = e[1]
    if node1 in A and node2 in A:
        d_i= G.degree[node1]
        d_j= G.degree[node2]
        Q += (1 - (d_i * d_j)/anfa)
    if node1 in B and node2 in B:
        d_i= G.degree[node1]
        d_j= G.degree[node2]
        Q += (1 - (d_i * d_j)/anfa)
Q/=anfa
print(f'Q: {Q}')
```

Output: Q: 0.265

3. Consider an undirected graph of seven nodes in the following figure. Calculate the edge betweenness of an edge (1,2). (10pt)



Solutions:

Edge betweenness: the number of shortest paths that pass along with the edge. To do that, we estimate the number of path starting from e_{12} to remaining nodes:

$$e_{12} \rightarrow 4 = \frac{1}{2}$$

$$e_{12} \rightarrow 5 = \frac{1}{2}$$

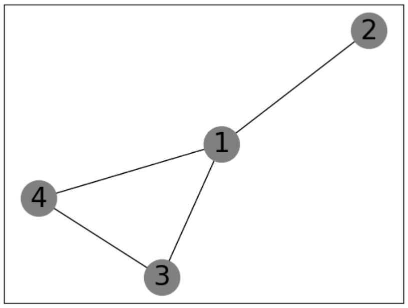
$$e_{12} \rightarrow 6 = \frac{1}{2}$$

$$e_{12} \rightarrow 7 = \frac{1}{2}$$

$$e_{12} \rightarrow 3 = 1$$

Edge betweenness = $\sum_{e_{12} \rightarrow 3,4,5,6,7} = 3$

4. Consider an undirected graph G of four nodes in the following figure. (10pt)



$$score(i,j) = \beta \tilde{A}_{ij} + \beta^2 \tilde{A}_{ij}^2 \qquad (2)$$

Where $\tilde{A}_{ij}$ is the element *(i, j)* in the normalized adjacency matrix of G, $\beta = 1$ is a parameter of the predictor.

a) Calculate the adjacency matrix A, the degree-normalized adjacency matrix $\tilde{A}$, and 2-step adjacency matrix $\tilde{A}^2$ of the graph G.

```
import numpy as np
A = nx.adjacency_matrix(G)
A = A.todense()
A
```

```
C:\Users\user\AppData\Local\Temp\ipyker
ray instead of a matrix in Networkx 3.0
  A = nx.adjacency_matrix(G)
```

```
matrix([[0, 1, 1, 1],
        [1, 0, 0, 0],
        [1, 0, 0, 1],
        [1, 0, 1, 0]], dtype=int32)
```

```
A_hat = np.zeros((4, 4))
for i, src in enumerate(G.nodes()):
    src_degree = G.degree(src)
    for j, dst in enumerate(G.nodes()):
        if G.has_edge(src, dst):
            A_hat[i][j] = round(1/src_degree,3)
A_hat
```

```
array([[0.   , 0.333, 0.333, 0.333],
       [1.   , 0.   , 0.   , 0.   ],
       [0.5  , 0.   , 0.   , 0.5  ],
       [0.5  , 0.   , 0.5  , 0.   ]])
```

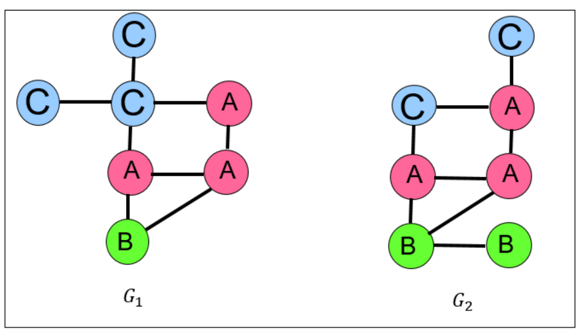
```
A_hat_2 = np.dot(A_hat,A_hat)
A_hat_2
```

```
array([[0.666 , 0.    , 0.1665, 0.1665],
       [0.    , 0.333 , 0.333 , 0.333 ],
       [0.25  , 0.1665, 0.4165, 0.1665],
       [0.25  , 0.1665, 0.1665, 0.4165]])
```

b) The Equation (2) presents the Katz score measurement between two nodes *(i, j)*. Apply the Equation (2) to calculate the Katz score between two nodes *(1, 2)*.

$$score(1,2) = \beta \tilde{A}_{12} + \beta^2 \tilde{A}_{12}^2 = 1*0.333 + 1*0 = 0.333$$

5. Calculate the graph edit distance between two graphs G_1 and G_2. The set of elementary graph edit operators includes: vertex insertion, vertex deletion, edge insertion, and edge deletion. In addition, the cost of deletion and insertion operators is 2 and 1, respectively. (5pt)

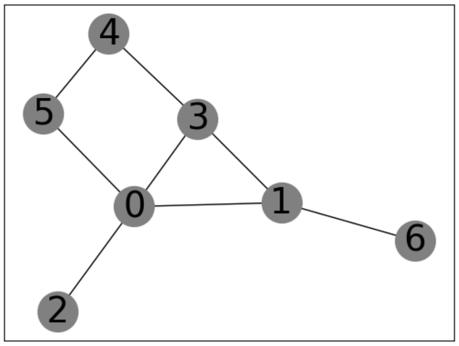


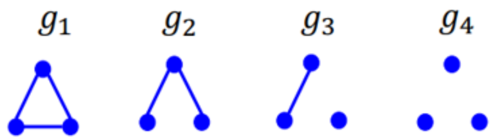
The cost of transforming from G_1 to G_2:

delete edge C-C: 2
delete node C: 2
delete edge C-C: 2
insert edge A-C: 1
insert node B: 1
insert edge B-B: 1.

Therefore, the graph edit distance: 9

6. Consider an undirected graph G of seven nodes in the following figure. There are four graphlets g_1, g_2, g_3, and g_4. (5pt)
   a) Count the number of the kernel sub-graphs of limited size 3.
   b) Make a feature vector for graph G based on these graphlet kernels.





SOLUTIONS:

a) Count subgraphs:

$N(g_1)$=1:

(0,1,3)

$N(g_2)$=11:

(0,1,6), (0,3,4), (0,5,4),
(1,3,4), (1,0,2), (1,0,5),
(2,0,5), (2,0,3),
(3,1,6), (3,0,5), (3,4,5)

$N(g_3)$=15

(0,1, 4), (0,2,4), (0,2,6), (0,3,6), (0,5,6),
(1,3,5), (1,3,2),
(1,6,2), (1,6,4), (1,6,5)
(3,4,2), (3,4,6),
(4,5,1), (4,5,2), (4,5,6),

$N(g_4)$=8

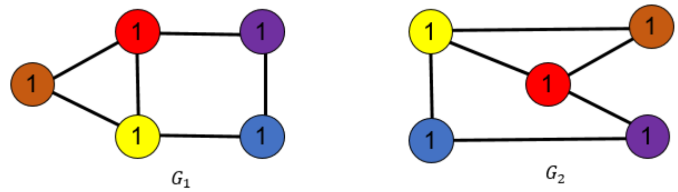
(0,4,6)
(1,2,5), (1,2,4)
(2,3,6), (2,3,5), (2,4,6), (2,5,6)
(3,5,6)

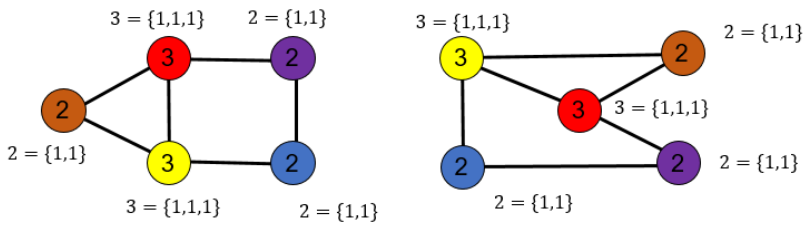
b) f_G = (1, 11,15,8)

7. Consider two undirected graphs in the following figure. (10pt)
   a) Conduct Weisfeiler-Lehman (WL) relabeling process with the maximum degree, 3. Then, using the Weisfeiler-Lehman isomorphism testing, determine whether two graphs are isomorphic or not?

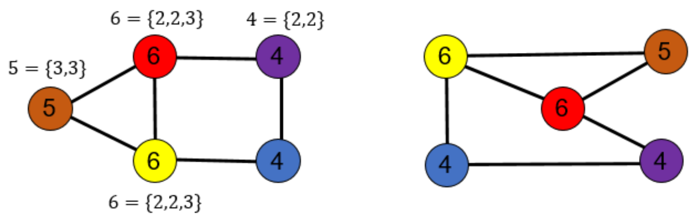
Two graphs are isomorphic:

Step 1: set node label =1 for all nodes

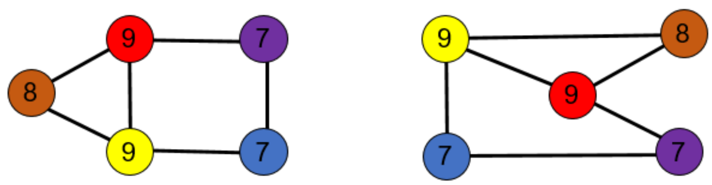




Step 2:



Step 3:

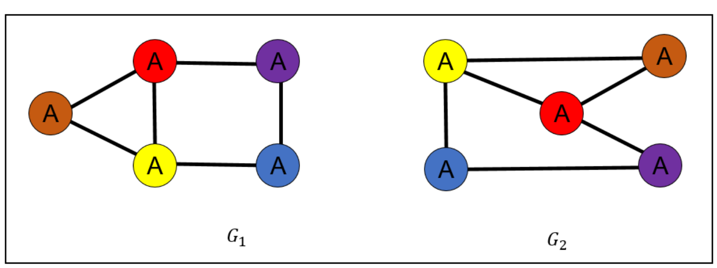


b) Make feature vectors for the graphs based on frequency of node degrees.

In G_1 , there are two nodes with degree 3, three nodes with degree 2, $f(G_1)=(2,3)$

In G_2 , there are two nodes with degree 3, three nodes with degree 2, $f(G_2)=(2,3)$

c) Make feature vectors for the graphs based on frequency of the WL subgraphs.



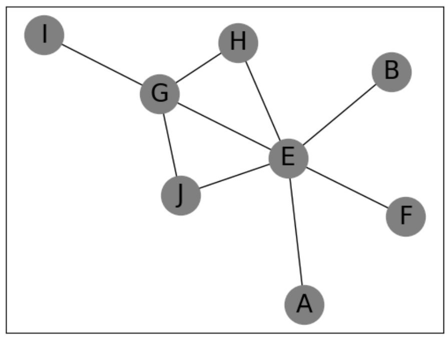
Consider 'B' as brown node, R: read node, Y: yellow node, P: Purple node, Bl: Blue node.

For G_1, we have types of the subgraphs:

Type 1: ['B', 'R'], ['B', 'Y'], ['R', 'Y'], ['R', 'P'], ['Bl', 'Y'], ['Bl', 'P']: 6
Type 2: ['B', 'R', 'Y']: 1
Type 3:['B', 'R', 'P'], ['B', 'Y', 'Bl'], ['R', 'Y', 'P'], ['R', 'Y', 'Bl'], ['R', 'P', 'Bl'], ['Y', 'P', 'Bl'] 6
Type 4: ['B', 'R', 'Y', 'P'], ['B', 'R', 'Y', 'Bl']: 2
Type 5: ['B', 'R', 'P', 'Bl'], ['B', 'Y', 'P', 'Bl']:2
Type 6: ['R', 'Y', 'P', 'Bl']: 1
Therefore, the feature vector: (1, 2, 2, 6, 1, 6)

Similarly, for G_2, we have the feature of the subgraphs: (1, 2, 2, 6, 1, 6)

8. Consider an undirected graph with eight nodes in the following figure. A biased random walk (Node2Vec algorithm) has the return parameter $p = 0.5$ and the in-out parameter $q = 0.5$. Assume that all edge weights of the graph are 1 and the walker is currently on node G by departing from node E. Calculate transition probabilities from node G to its neighbors. (10pt)



$$P_{G\to I} = 1\times\frac{1}{q} = \frac{1}{0.5} = 2$$
$$P_{G\to J} = 1\times 1 = 1$$
$$P_{G\to H} = 1\times 1 = 1$$
$$P_{G\to E} = 1\times\frac{1}{p} = \frac{1}{0.5} = 2$$

9. Write a python function to compute the degree-normalized adjacency matrix. (10pt)

```
#Input: a dense adjacency A.
```

`#Output: X, a degree-normalized adjacency matrix (`$\tilde{A} = D^{-1}A$`, where` *D* `is the degree matrix of the graph)`
`#Note: Students can only use Python code (without using inbuilt functions, such as min, max, sum, etc.). In the NetworkX library, students can use functions 'degree()', 'nodes()', 'has_edge()'.`

```
def Normalized(A):
    for i in range(A.shape[0]):
        sum_d = 0
        for j in range(A.shape[0]):
            sum_d += A[i,j]
        if sum_d >0:
            for j in range(A.shape[0]):
                A[i,j] = A[i,j]/sum_d
    return A
```

10. The bellow function is designed to calculate an Eigenvector centrality for a given graph. Write codes to fill the blank "YOUR CODE HERE". (20pt)
    a. Complete the code to measure the Katz centrality.
    b. Complete the code to measure the PageRank centrality.

```
#Input:
#G: A networkx graph.
#max_iter: integer, maximum number of iterations.
#tol: float, error to check convergence.
#nstart: dictionary, starting value of eigenvector iteration.
#weight: None or string, all edge weights are considered equal.
#alpha: float, attenuation factor
#alpha_pg: float, damping parameter for PageRank, default=0.85.
#beta: scalar, (default=1.0), controls the initial centrality
#Output:
#nodes: dictionary, Dictionary of nodes with centralities as the value.
```

```
def Eigenvector(G,alpha=0.1,beta=1.0,max_iter=100,tol=1e-4, nstart,
weight, alpha_pg):
    if len(G) == 0:
      print ("cannot compute centrality for the null graph")
    # If no initial vector is provided, start with the all-ones
vector.
    if nstart is None:
          nstart = {v: 1 for v in G}
    if all(v == 0 for v in nstart.values()):
          print("initial vector cannot have all zero values")
    nstart_sum = sum(nstart.values())
    x = {k: v / nstart_sum for k, v in nstart.items()}
    nnodes = G.number_of_nodes()
    # <For page_rank information>
    D = G.to_directed()
    # Create a copy in (right) stochastic form
    W = nx.stochastic_graph(D, weight=weight)
    # Assign uniform personalization vector if not given
    dangling_weights = dict.fromkeys(W, 1.0 / N)
    dangling_nodes = [n for n in W if W.out_degree(n, weight)== 0.0]
    # </For page_rank information>
    for _ in range(max_iter):
        xlast = x
        x = xlast.copy()
        danglesum = alpha_pg * sum(xlast[n] for n in dangling_nodes)
        # Start with xlast times I to iterate with (A+I)
        # do the multiplication y^T = x^T A (left eigenvector)
        for n in x:
            for nbr in G[n]:
                w = G[n][nbr].get(weight, 1) if weight else 1
                x[nbr] += xlast[n] * w
         #YOUR CODE HERE
          #KATZ
        for n in x:
            x[n] = alpha * x[n] + beta

        #Pagerank
                #x[nbr] += xlast[n] * w
                x[nbr] += alpha * xlast[n] * w
        for n in x:
            x[n] += danglesum * p.get(n, 0) + (1.0 - alpha) *
p.get(n, 0)
        norm = math.hypot(*x.values()) or 1
        x = {k: v / norm for k, v in x.items()}
        if sum(abs(x[n] - xlast[n]) for n in x) < nnodes * tol:
            return x
```