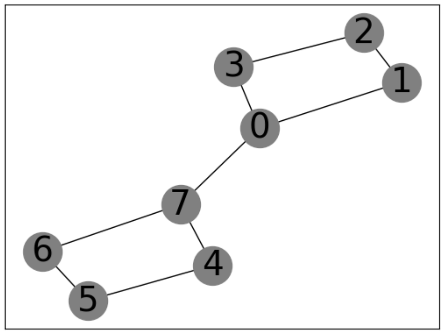
Final Exam (Graph Mining – Spring 2023)

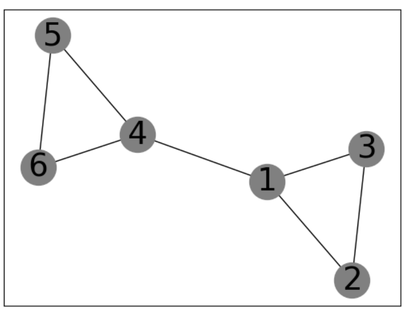
Full Name:
Student ID:

- 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)



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)

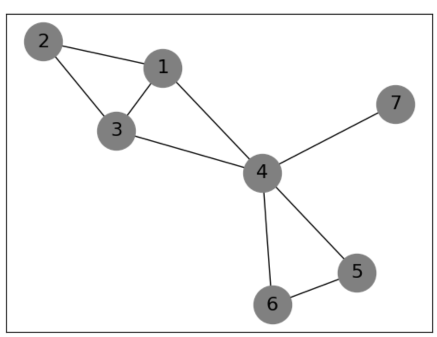


$$Q = \frac{1}{2m}\sum_{i,j}\left(A_{ij} - \frac{d_i d_j}{2m}\right)\cdot\delta\left(v_i, v_j\right) \tag{1}$$

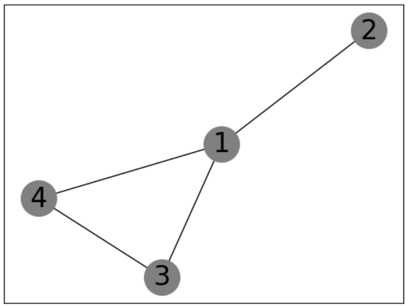
$$\delta\left(v_i, v_j\right) = \begin{cases} 1 & \text{if } v_i \text{ and } v_j \text{ are in the same community.} \\ 0 & \text{otherwise.} \end{cases}$$

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

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



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

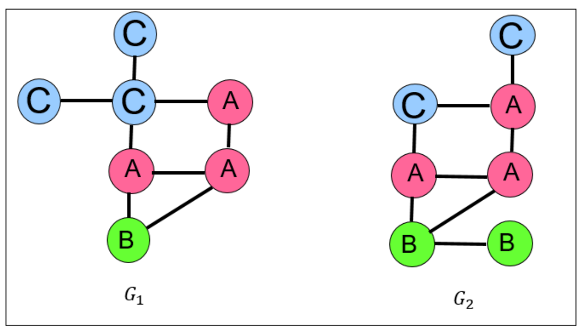


$$score(i, j) = \beta\tilde{A}_{ij} + \beta^2\tilde{A}_{ij}^2 \qquad (2)$$

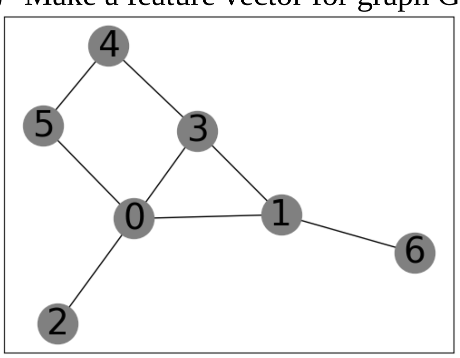
Where $\tilde{A}_{ij}$ is the element *(i, j)* in the normalized adjacency matrix of G, $\beta = 1$ is a parameter of the predictor.

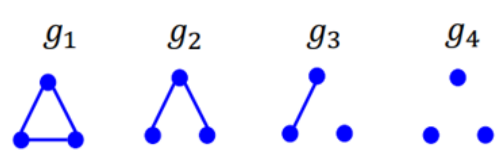
a) Calculate the adjacency matrix A, the degree-normalized adjacency matrix $\tilde{A}$, and 2-step adjacency matrix $\tilde{A}^2$ of the graph G.
b) The Equation (2) presents the Katz score measurement between two nodes *(i, j)*. Apply the Equation (2) to calculate the Katz score between two nodes *(1, 2)*.

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)

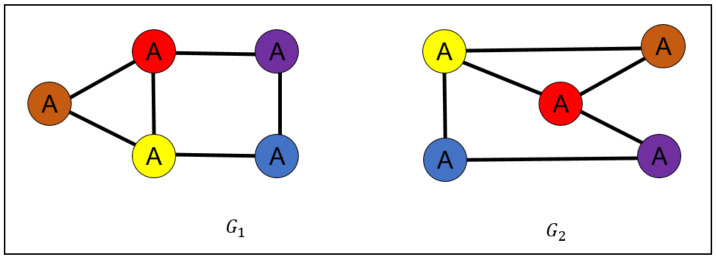


6. Consider an undirected graph G of seven nodes in the following figure. There are four graphlets g_1, g_2, g_3, and g_4. (5pt)
a) Count the number of the kernel sub-graphs of limited size 3.
b) Make a feature vector for graph G based on these graphlet kernels.

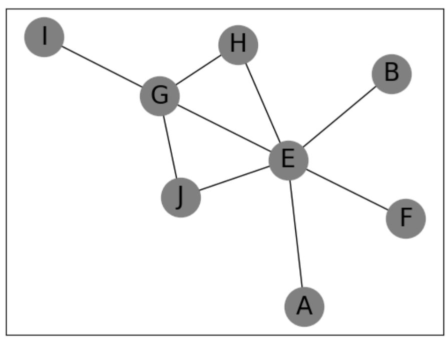




7. Consider two undirected graphs in the following figure. (10pt)
   a) Conduct Weisfeiler-Lehman (WL) relabeling process with the maximum degree, 3. Then, using the Weisfeiler-Lehman isomorphism testing, determine whether two graphs are isomorphic or not?
   b) Make feature vectors for the graphs based on frequency of node degrees.
   c) Make feature vectors for the graphs based on frequency of the WL subgraphs.



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)



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 (Ã=D⁻¹A, 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):
    #YOUR CODE HERE.
    return X
```

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
        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
```