**Exercise 1**
Implement the following game-theoretic centrality measures:
1) **shapley_degree**: this is the Shapley value for the characteristic function

$$\textbf{value(C)} = |\mathbf{C}| + |\mathbf{N(C)}|,$$

where **N(C)** is the set of nodes outside **C** with at least one neighbor in **C**;
2) **shapley_threshold(k)**: this is the Shapley value for the characteristic function

$$\textbf{value(C)} = |\mathbf{C}| + |\mathbf{N(C,k)}|,$$

where **N(C,k)** is the set of nodes outside **C** with at least **k** neighbors in **C**;
3) **shapley_closeness**: this is the Shapley value for the characteristic function

$$\textbf{value(C)} = \Sigma_u \textbf{ dist(u,C)},$$

where **dist(u,C)** is the minimum distance between **u** and a node of **C**.

Recall that the naive implementation of Shapley value requires a running time that is exponential in the number of nodes of the network. You are instead required to provide a polynomial time algorithm for the above measures. On the e-learning platform you fill find material that will help you in designing and implementing these algorithms.

Implement also the Friedkin-Johnsen (FJ) dynamics, that works as follows:

- each node **u** has a private belief $\mathbf{b_u}$ in **[0, 1]** and a stubbornness value $\mathbf{s_u}$ in **[0,1]**;
- at each time step **t** each node publicizes an opinion $\mathbf{x_u(t)}$ in **[0,1]** where
  - $\mathbf{x_u(0) = b_u}$, i.e., the initial opinion is exactly its belief;
  - $\mathbf{x_u(t) = s_u\ b_u + (1\text{-}s_u)\ sum_{v\ in\ N(u)}\ 1/N(u)\ x_v(t\text{-}1)}$, i.e., the opinion at time **t** is a weighted average of the private belief and of the opinion publicized by its neighbors at the previous step.

Does this dynamics converge to a stable state (i.e., a state in which no agent update her opinion – you may assume a finite precision for opinion of at most 5 decimal digits)? Provide either a formal proof or experimental evidence for your answer.

**Exercise 2**
Consider the network **N** represented in the file *net_x*, that has been generated with one of the network models seen during the course.

You have to analyze the network **N** and guess which model has been used for creating it.
Your guess has to be supported by an appropriate set of experiments to confirm that networks generated with the proposed model have characteristics similar to **N** (note that you have to guess also the parameters of the model).

During the discussion of the project, you will be asked to motivate your guess. Motivations may be related to both theoretical properties of the models seen during the course (e.g., "I analyzed the provided network and I observed that its node degree distribution follows a power law. Hence, I conclude that it is not possible that the graph has been generated with a model **random(n,p)**."), and to experimental evidence (e.g., "I generated a lot of random graphs with **p = 1/3**, and none of them had similar properties as the provided network. Hence I conclude that it is improbable that the graph is **random(n, 1/3)**").

A bonus point will be assigned to all the components of the groups that guessed correctly the model (and parameters) used to generate **N**.

**Exercise 3**
Suppose there is an election and the voters are connected through a social network. **G = (V, E)**. Suppose tha there are **n** voters, represented by the nodes of the graph G, and **m** candidates. Each

candidate **i** has a position $\mathbf{p_i}$ in **[0,1]** that represents her political tendency (for example, a candidate whose position is close to **0** or **1** is, respectively, an extreme-left or an extreme right candidate, while a candidate with position close to **1/2** is moderate).
Each voter **u** has *single-peaked* preferences with peak in $\mathbf{b_u}$, i.e., she ranks candidates according to the distance of their positions from $\mathbf{b_u}$, by breaking possible ties in favor of the candidate on lthe left of $\mathbf{b_u}$ (thus, the most preferred candidate is the one whose position is closest to $\mathbf{b_u}$, the second most preferred candidate is the one with the second closest position and so on).

The election occurs according to a plurality voting rule (see lesson about voting for a definition). We call an election *truthful* if each voter **u** votes for the candidate closest to her peak $\mathbf{b_u}$.

On the other hand, a voter can be influenced by opinion campaigns run over the social network and she could be induced to vote a candidate different from her favorite one.
Specifically, we consider a manipulator that wants to improve the outcome of a given candidate **c**. To this aim the manipulator can select at most **B** voters (in the following called *seeds*), alter their peaks and use their influence to induce a change in the votes expressed by other voters.

We assume that the voting opinions diffuse over the network according to a FJ dynamics. Specifically, if **S** is the set of at most **B** seeds chosen by the manipulator, then

1. Set $\mathbf{x_u(0) = b_u}$, and $\mathbf{s_u=1/2}$ for every **u** not in **S**
2. For **u** in **S**, let $\mathbf{b_u = b'_u}$, where $\mathbf{b'_u}$ is defined by the manipulator, and set $\mathbf{x_u(0) = b_u}$, and $\mathbf{s_u = 1}$
3. Run the FJ dynamics with this configuration
4. Once the dynamics reaches the equilibrium at time step **t**, update the preferences of voters by setting the peak $\mathbf{p_u = x_u(t)}$
5. Re-run the election with voter's peaks in $\mathbf{p_u}$. We call this election *manipulated*.

You have to design an algorithm that, given a network **G**, a set of **m** candidates with their positions $\mathbf{(p_1, \ldots, p_m)}$, a special candidate **c**, a budget **B**, and the initial peaks of all the voters $\mathbf{(b_1, \ldots, b_n)}$, returns a set S of at most **B** seeds and a peak value $\mathbf{b'_u}$ for each seed **u** in **S**, such that the difference between the number of votes obtained by the candidate **c** in the manipulated election and the truthful one is maximized.

All the proposed manipulation algorithms will be tested on a common input. The group providing the larger increment in the number of votes for the candidate **c** will receive a bonus point.

INSTRUCTION FOR THE SUBMISSION:
Your code must include a function **manipulation(G, p, c, B, b)**, where **G** is an undirected, unweighted graph, **p** is a Python list with each element in **[0,1]**, **c** is in **{0, …, len(p)-1}**, **B** is a positive integer, and **b** is a Python list such that **len(b) = len(G.nodes())** with each element in **[0,1]**. The function must print only one string that contains the following three elements separated by a comma:

- the group number;
- the number of votes for candidate **c** before the manipulation occurs;
- the number of votes for candidate **c** after the manipulation.