Due: 22^{nd} **March 2020** **Total: 100 points**

Instructions

- Institute Plagiarism Policy applies to all HWs.
- It has been made explicit wherever usage of `networkx` or `snap.py` is required. In all other cases, these libraries can only be used for loading and representing the graphs. The implementations need to be done from scratch.
- **Submission Instructions:**
  - All the submissions must be inside a `zip` file named `a3_<your_roll_number>.zip` containing a report named `report.pdf` and a folder named `src` containing all your scripts.
  - All the code must be well documented. Failure to do so will result in a deduction of 2% of the total from the obtained marks in the respective question.
  - All the plots and analyses required should be uploaded in a `PDF` file named `report.pdf`.

Problem 1: Decision Based Cascades: A Local Election (50 points)

- It's election season and two candidates, Candidate `A` and Candidate `B`, are in a hotly contested city council race in sunny New District. You are a strategic advisor for Candidate `A` in charge of election forecasting and voter acquisition tactics.
- Based on careful modeling, you've created two possible versions of the social graph of voters. Each graph has $10,000$ nodes, each denoted by an integer id between 0 and 9999. The edge list for Graph 1 is in `g1.edgelist` and for Graph 2 is in `g2.edgelist`. Both the graphs are **undirected**.
- Now, the people of New District are extremely partisan. In fact, 40% of them have made up their mind that they will vote for `A` and 40% have decided to vote for `B`. 20% of the remaining population is still undecided.
- Each node's support is determined by the last digit of their id. If the last digit is 0-3, the node supports `B`. If the last digit is 4-7, the node supports `A`. And if the last digit is 8 or 9, the node is undecided.
- The undecided voters will go through a 7-day decision period where they choose a candidate based on the majority of their friends. The decision period works as follows:
  - The graphs are initialized with every voter's initial state.
  - In each iteration, every undecided voter is assigned a vote. If the majority of their friends support `A`, they now support `A`. If the majority of their friends support `B`, they now support `B`. "Majority" for `A` means that more of their friends support `A` than the number of their friends supporting `B`, and vice versa for `B`.
  - If they have an equal number of friends supporting `A` and `B`, we assign `A` or `B` in alternating fashion, starting with `A`. This alternation happens at a global level for the whole network, across all rounds, so we break the first tie with `A`, the second with `B`, and so on (not on a per node basis). Keep a single global variable that keeps track of whether the current alternating vote is `A` or `B`, and initialize it to `A` in the first round. Then as you iterate over nodes in order of increasing ID, whenever you assign a vote using this alternating variable, change its value afterwards.
  - When updating the votes, use the values from the current iteration. So, for example, when updating the votes for node 10, you should use the votes for nodes 0-9 from the current iteration, and nodes 11 and onwards from the previous iteration.

- The process described above happens 7 times.
- On the 8^{th} day, it's election day, and the votes are counted.

Note: Only the undecided voters go through the decision process. The decision process does not change the loyalties of those voters who have already made up their minds.

Problem 1a: Basic setup and forecasting (10 points)

Read in the two graphs using software of your choice. Assign initial vote configurations to the network. Then, simulate the 7 day voting process. Which candidate wins in Graph 1, and by how many votes? Which candidate wins in Graph 2, and by how many votes?

Problem 1b: TV Advertising (16 points)

Through donations from your loyal supporters, you have amassed a sum of ₹ 90,000. You have decided to spend this money on sending out TV advertisements on a popular local TV news channel. The problem is that only 100 of the residents, those with ids 3000-3099 watch this channel. For each ₹ 10,000 you spend, you reach 10 additional voters, starting with 3000-3009. But the catch is that anyone who watches the advertisement, no matter what their previous alignment, decides to switch their vote to `A`.

In accordance with the election rules, the advertisements happen before the decision period.

- Now, you will simulate the effect of advertising spending on the two possible social graphs.
- First, you'll read in the two graphs again, and assign the initial configurations as before.
- But now, before the decision process, you will run your ₹`k` worth of ads, then you will go through the decision process of counting votes.

For each of the two social graphs, please plot `k` (the amount you spend) on the x-axis (for values $10000, 20000, \ldots, 90000$) and the number of votes you win by on the y-axis (this is a negative number if you lose). What's the minimum amount you can spend to win the election in each of the two social graphs?

Note: The TV advertising effects all the voters and not just those who are undecided.

Problem 1c: Dining with the big shots (16 points)

Influential people can have a huge impact in swaying popular opinion towards a brand or product. This is especially true in smaller communities. But for this, you need to really make the influential people believe in your party's platform. To do this, you make a list of celebrities who hail from New District, and sort them by popularity. (In this case, popularity is given by the degree of the node in the network. In case of degree ties, choose the node with the lower ID first.)

Your idea is to invite these influential people to a campaign dinner, organised at the best restaurant in New District, "*The Social Network*". Only problem is that each plate there costs ₹ 10000. However, anyone who attends this dinner is immediately swayed to vote for `A`, no matter their previous decision.

Once again, the dinner is held before the decision period, where you can spend ₹`k` and then the decision period is simulated.

For each of the two social graphs, please plot `k` (the amount you spend) on the x-axis (for values 10000, 20000, ..., 90000) and the number of votes you win by on the y-axis (this is a negative number if you lose). What's the minimum amount you can spend to win the election in each of the two social graphs?

Note: All voters invited to the dinner are swayed to vote for `A` and not just those who are undecided.

Problem 1d: Analysis **(8 points)**

Briefly (1-2 sentences) explain some property (eg. degree distribution, clustering coefficient, etc.) of the 2 graphs that can explain the results of the last 2 questions. Also, explain how you'll spend your ₹ 90,000 and why (there's no specific answer we're looking for here, but the strategy you use must be a reasonable strategy and you must explain it in 1-2 sentences. Strategies like, "*I will spend the entire money on TV ads*" is an example of an unreasonable strategy.).

Problem 2: Approximating edge betweenness (50 points)

In class we saw how Betweenness Centrality of an edge is a central concept in some Community Detection Techniques, like the Girvan–Newman algorithm. In this question, we will take a deeper look at this concept.

Given an undirected graph $G = (V, E)$, the betweenness centrality of an edge $\{u, v\} \in E$ is defined as:

$$B(\{u,v\}) = \sum_{\{s,t\} \in V \times V} \frac{\sigma_{st}(\{u,v\})}{\sigma_{st}}$$

Here, σ_{st} is the total number of shortest paths between s and t, and $\sigma_{st}(\{u, v\})$ is the number of shortest paths between s and t that contain the edge $\{u, v\}$.

Algorithm 1: Exact Betweenness Centrality

- For each vertex $s \in V$, perform a BFS from s, which induces a BFS tree T_s.
- For each $v \in V$, let v's parent set $P_s(v)$ be defined as the set of nodes $u \in V$ that precede v on some shortest path from s to v in G.
- During the BFS, also compute, for every $v \in V$, the number σ_{sv} of shortest paths between s and v, according to the recurrence:

$$\sigma_{sv} = \begin{cases} 1, & \text{if } v = s \\ \sum_{u \in P_s(v)} \sigma_{su}, & \text{otherwise} \end{cases}$$

- After the BFS has finished, compute the so-called dependency of s on each edge $\{u, v\} \in E$ using the recurrence:

$$\delta_s(\{u,v\}) = \begin{cases} \frac{\sigma_{su}}{\sigma_{sv}}, & \text{if } v \text{ is a leaf node of } T_s \\ \frac{\sigma_{su}}{\sigma_{sv}}\left(1 + \sum_{x : v \in P_s(x)} \delta_s(\{v,x\})\right), & \text{otherwise} \end{cases}$$

- For the purpose of definition, assume without loss of generality that the shortest path from s to u is shorter than to v.
- Finally, the betweenness centrality of $\{u, v\}$ equals $B(\{u, v\}) = \sum_{s \in V} \delta_s(\{u, v\})$

Cons of Algorithm 1:

- Algorithm 1 has a time complexity of $O(nm)$, where n and m are the numbers of nodes and edges, respectively, and there is no faster known algorithm for computing betweenness centrality exactly in undirected, unweighted graphs.
- Also, using Algorithm 1, one needs to compute the betweenness centrality of all edges, even if one is interested only in that of some select edges.

In this problem you'll explore an approximate version of Algorithm 1. It is nearly identical to Algorithm 1, with the difference that it doesn't start a BFS from every node but rather samples starting nodes randomly with replacement. Also, it can be run for any edge $e \in E$ independently, without necessarily having to compute the centrality of all other edges as well.

Algorithm 2: Approximate Betweenness Centrality

- Repeatedly sample a vertex $v_i \in V$
- perform a BFS from v_i (as in Algorithm 1) and maintain a running sum Δ_e of the dependency scores $\delta_{v_i}(e)$ (one Δ_e for each edge e you're interested in).
- Sample until Δ_e is greater than cn for some constant $c \geq 2$. Let the total number of samples be k.
- The estimated betweenness centrality score of e is given by $\frac{n}{k}\Delta_e$

This is much faster than Algorithm 1.

Simulation

- Generate a random graph following the Barabási–Albert preferential attachment model, on $n = 1000$ nodes and attaching each node to 4 existing nodes when adding it to the network. (`barabasi_albert_graph(1000, 4)` in NetworkX) (5 points)
- Compute the exact betweenness centrality for all edges $e \in E$ using Algorithm 1. (Implement it yourself.) (20 points)
- Compute the approximate betweenness centrality for all edges $e \in E$ using Algorithm 2. Use $c = 5$ and sample at most $\frac{n}{10}$ random starting nodes v_i. (20 points)
- Each algorithm induces an ordering over edges with respect to betweenness centrality. Hand in the following plot: One curve for Algorithm 1, one for Algorithm 2, overlaid in the same plot. If the edge with the x^{th} largest betweenness centrality (according to the respective algorithm) has betweenness centrality y, draw a dot with co-ordinates (x, y). Please use a logarithmic y-axis. Comment very briefly on the curves. (5 points)