**Due:** $28^{th}$ **February 2020** **Total: 50 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 `a1_<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: Generating Barabási-Albert Networks (30 points)

- `You can make use of networkx, scipy, numpy etc. for this question`
- Generate a network with $N = 10^4$ nodes using the Barabási-Albert model with $m = 4$. Use as initial condition a fully connected network with $m = 4$ nodes.
- Measure the degree distribution at intermediate steps, namely when the network has $10^2$, $10^3$ and $10^4$ nodes.
- Compare the distributions at these intermediate steps by plotting them together and fitting each to a power-law with degree exponent $\gamma$. Do the distributions "converge"? (For fitting the power law, you can take log and perform regression to get the power law parameter, or use any method for fitting a distribution that you find convenient. Libraries are allowed for this question.)
- Plot together the cumulative degree distributions at intermediate steps.
- Measure the average clustering coefficient in function of $N$.
- Measure the degree dynamics of one of the initial nodes and of the nodes added to the network at time $t = 100$, $t = 1,000$ and $t = 5,000$. For what analysis you have to do, take a look at Image 5.6 at this link.

## Problem 2: Implementing PageRank (20 points)

In this problem, you will learn how to implement the PageRank algorithm. You will be experimenting with a small randomly generated graph (assume graph has no dead- ends) provided in the `graph.txt` file.

It has $n = 100$ nodes (numbered $1, 2, \ldots, 100$), and $m = 1024$ edges, 100 of which form a directed cycle (through all the nodes) which ensures that the graph is connected. It is easy to see that the existence of such a cycle ensures that there are no dead ends in the graph. There may be multiple edges between a pair of nodes, your program should handle these instead of ignoring them. The first column in `graph.txt` refers to the source node, and the second column refers to the destination node.

Assume the directed graph $G = (V, E)$ has $n$ nodes (numbered $1, 2, \ldots, n$) and $m$ edges, all nodes have positive out-degree, and $M = [M_{ji}]_{n\times n}$ is a an $n \times n$ matrix as defined in class such that for any $i, j \in [1, n]$:

$$M_{ij} \begin{cases} \frac{1}{\deg(i)}, & \text{if } (i \longrightarrow j) \in E \\ 0, & \text{otherwise} \end{cases}$$

Here, $\deg(i)$ is the number of outgoing edges of node $i$ in $G$. By the definition of PageRank, assuming $1\beta$ to be the teleport probability, and denoting the PageRank vector by the column vector $r$, we have the following equation:

$$r = \frac{1-\beta}{n}\mathbf{1} + \beta M r$$

where, $\mathbf{1}$ denotes $n \times 1$ column vector.

Based on this, the PageRank algorithm runs as follows:

1. Initialise $r^{(0)} = \frac{1}{n}\mathbf{1}$
2. For $i$ from 1 to $k$, iterate: $r^{(i)} = \frac{1-\beta}{n}\mathbf{1} + \beta M r^{(i-1)}$

Run the aforementioned iterative process for 40 iterations (assuming $\beta = 0.8$) and obtain the PageRank vector $\mathbf{r}$. Compute the following:

- List the top 5 node ids with the highest PageRank scores.
- List the bottom 5 node ids with the lowest PageRank scores.