How to Stop a Pandemic

> **Topics:** Graphs, Social network analysis, Epidemiology, Testing

> **Due Date:** April 24 @ 11:59pm

In this assignment, we will use a graph to simulate the spread of a pathogen across a population. We will explore three different strategies to slow down or stop the spread of the pathogen. We provided two datasets that you can use to test your code. Feel free to create your own dataset.

Starter files here.

Step 1: Load the Dataset

Create a class named `InformationSpread.Java` and a test class named `InformationSpreadTest.Java`.

Implement and test `loadGraphFromDataSet`. This method will return the number of nodes processed by the method.

Notes:

- The first line of the dataset contains the number of vertices and edges (one direction) separated by a space. The provided graph implementations expect a directed graph; however, we are working with undirected graphs. Do not forget to add both directions of each edge to the graph! This will result in the total number of edges in the graph being twice the number reported in the file.
- The rest of the file consists of lines in the format `first second weight`, representing an edge from vertex `first` to vertex `second` with weight `weight`.
- There is no vertex with id 0 in the dataset. Take that into consideration when initializing the graph object. We will use vertex 0 to represent a vertex with degree 0. Those vertices will be listed in the file with a link to 0 (e.g. the line starting with `5 0` means that node 5 has no neighbors)

- The choice of the graph implementation (GraphL or GraphM) to use is up to you
- In this assignment, there is a possible transmission only if the weight of the edge is $\geq \tau$ (tau). You can ignore the edges with weight $< \tau$.
- The graph implementations expect an integer value as weight, so multiply the weights by 100 and cast to an `int`.
- The method returns the number of nodes **with at least one edge** contained in graph object. This number can be different from the total number of nodes in the dataset.

Also, implement and test `getNeighbors`.

> **WHY ARE WE IGNORING EDGES WITH WEIGHT $< tau$?**
> The weight of an edge represents the transmission probability between two individuals. In the real world, connections between people may have a higher transmission probability (say those with weakened immune systems) or a lower probability. For the simplicity of the assignment, and to reduce runtimes in large graphs, we will not include the edge in the graph if its weight is < 0.55 since transmission is unlikely. Thus, the nodes will be added to the graph but **not** counted by `loadGraphFromDataSet`.

Step 2: Transmission path

We want to know how the pathogen spreads between two nodes in the network. Implement and test `path`. The method takes two nodes and returns a collection containing all the nodes along the path from the source to the destination that has the highest probability of infection. Include the source and the destination in your collection. Return an empty Collection if there is no path from source to destination.

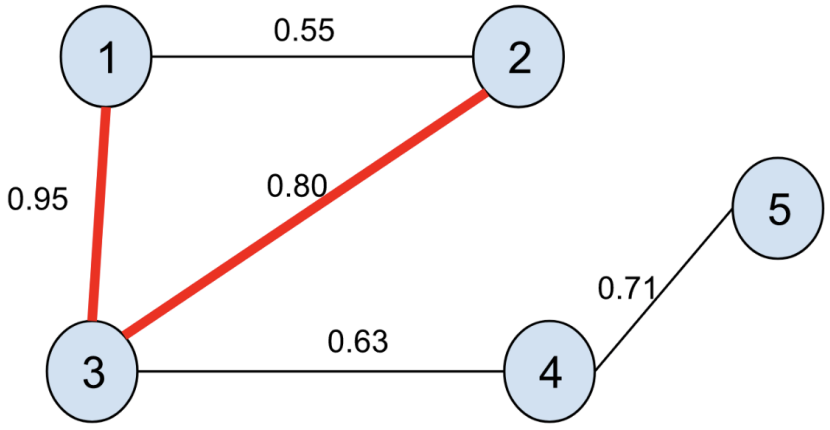
In order to compute the highest probability transmission path, we will perform a transformation f on the graph's weights like so: $f(w_{i,j}) = -\log w_{i,j}$. We assume that weights have been divided by `100` to convert the `int` weights back to probability values in the range $[0, 1]$. We take the negative of the log so that the shortest path using transformed weights $f(w)$ reflects the path of highest probability. The proof follows:

All pairwise infections are assumed to be independent of each other. Therefore, the *transmission probability* of an infection travelling from vertex s to vertex t along path $p_{s \to t}$ is assumed to be the $\Pi_{w_{i,j} \in p_{s,t}} w_{i,j}$. We seek the path $p^*_{s \to t}$ that has the highest probability of transmission among all paths from s to t.

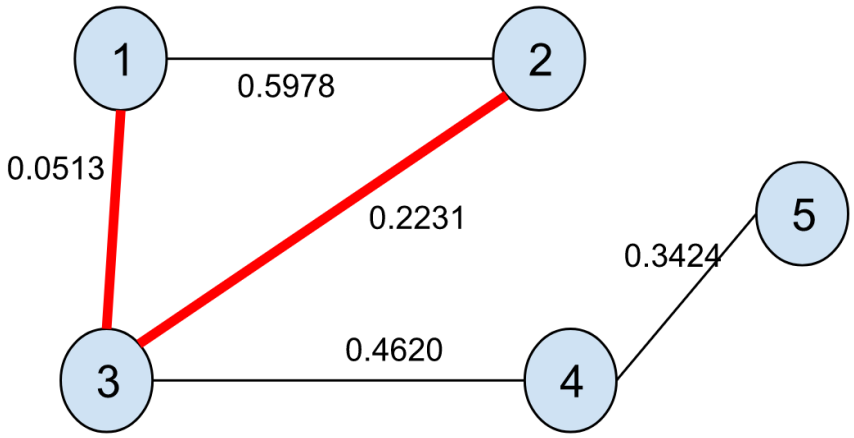
$$
\begin{aligned}
p^*_{s\to t} &= arg\,max_{p_{s\to t}} \prod_{w_{i,j}\in p_{s,t}} w_{i,j} \\
&= arg\,max_{p_{s\to t}} \log \prod_{w_{i,j}\in p_{s,t}} w_{i,j} \\
&= arg\,max_{p_{s\to t}} \sum_{w_{i,j}\in p_{s,t}} \log w_{i,j} \\
&= arg\,min_{p_{s\to t}} - \sum_{w_{i,j}\in p_{s,t}} \log w_{i,j} \\
&= arg\,min_{p_{s\to t}} \sum_{w_{i,j}\in p_{s,t}} - \log w_{i,j} \\
&= arg\,min_{p_{s\to t}} \sum_{w_{i,j}\in p_{s,t}} f(w_{i,j})
\end{aligned}
$$

Therefore, to find the path with the highest transmission probability, it is sufficient to find the shortest path in the transformed graph. Dijkstra's algorithm is acceptable here since all weights will be positive values: the log of a probability value is at most 0, so negating all logs gives non-negative edge weights.

For example, given the following graph,



we can compute the transformed graph:



and observe that the path with the highest transmission probability passes through `3` between `1` and `2`. This should make some intuitive sense: a transmission between `1` and `3` and between `3` and `2` are both very likely, with a combined probability of 76%, whereas the transmission from `1` to `2` directly has only a 55% probability of occurring.

Step 3: Basic Reproduction Number

"The basic reproduction number, R0, is defined as the expected number of secondary cases produced by a single (typical) infection in a completely susceptible population."

$$R0 = \tau c d$$

where:

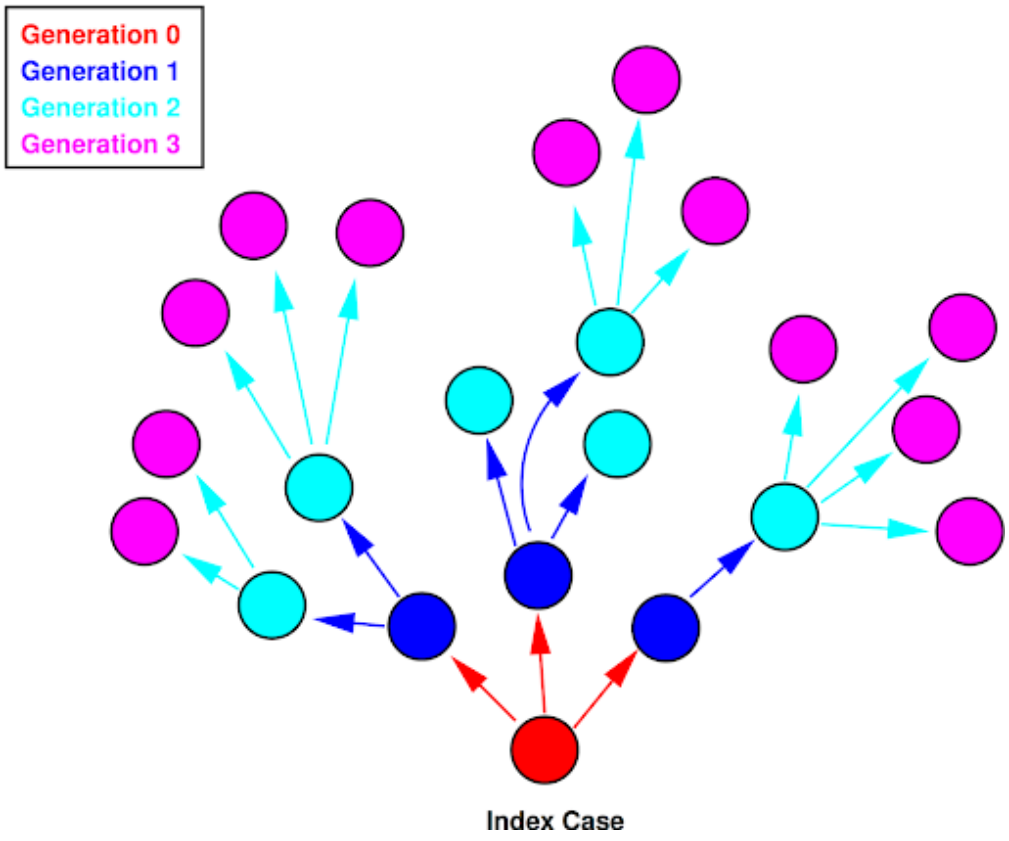
- τ is the transmissibility (i.e., probability of infection given contact between a susceptible and infected individual). This value was passed in when calling `loadGraphFromDataSet`.
- c is the average rate of contact between susceptible and infected individuals, it is the **average degree of the graph**
- and d is the duration of infectiousness. In our case, we will assume immediate infectiousness, $d = 1$.

Implement and test `avgDegree` which computes the average degree of the graph, and `rNumber` which computes the basic reproduction number using the formula above with the provided constants.

Step 4: Generations

"Generations in epidemic models are the waves of secondary infection that flow from each previous infection. So, the first generation of an epidemic is all the secondary infections that

result from infectious contact with the index case, who is of generation zero."



In this task, we will explore how many generations are needed in order for a pathogen to infect a given proportion of the population. Given a starting node or index case (we will call it seed), we will count how many generations it will take for the pathogen to go from seed to a specific percentage of the nodes in the population. In this context, we are assuming that edges represent deterministic infections—there is no probability associated with transmission here.

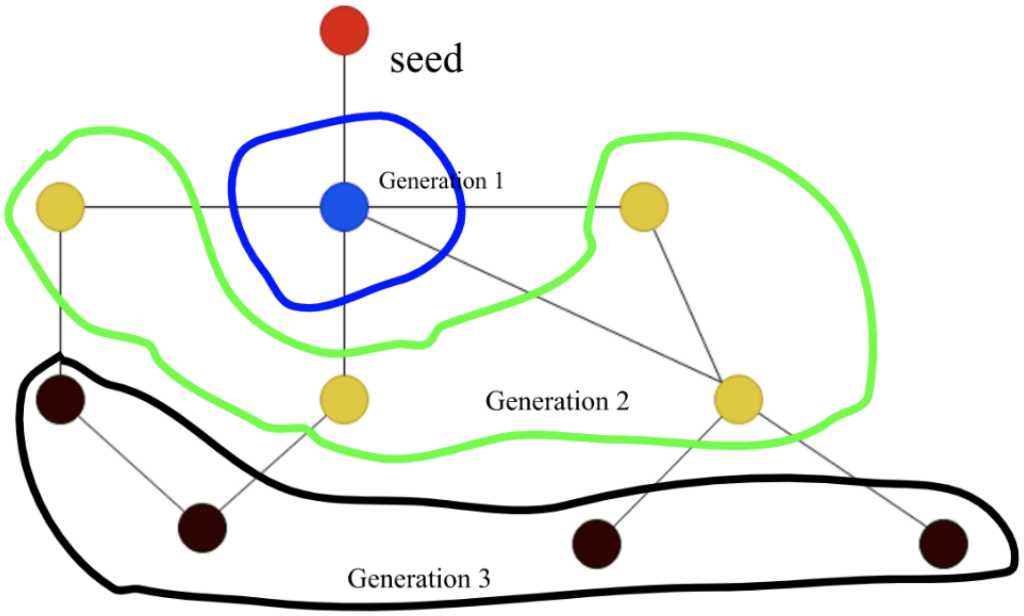
A generation is computed as follows:

- The seed is infected during generation 0
- All the neighbors of the seed are in generation 1
- All the neighbors of vertices in generation 1 that are not in generation 0 or generation 1 are in generation 2
- All the neighbors of vertices in generation 2 that are not in generations 0, 1, or 2 are in generation 3, and so on.
- We stop when we have reached a given percentage of nodes.

Given the following graph and a seed represented by the vertex colored in red,

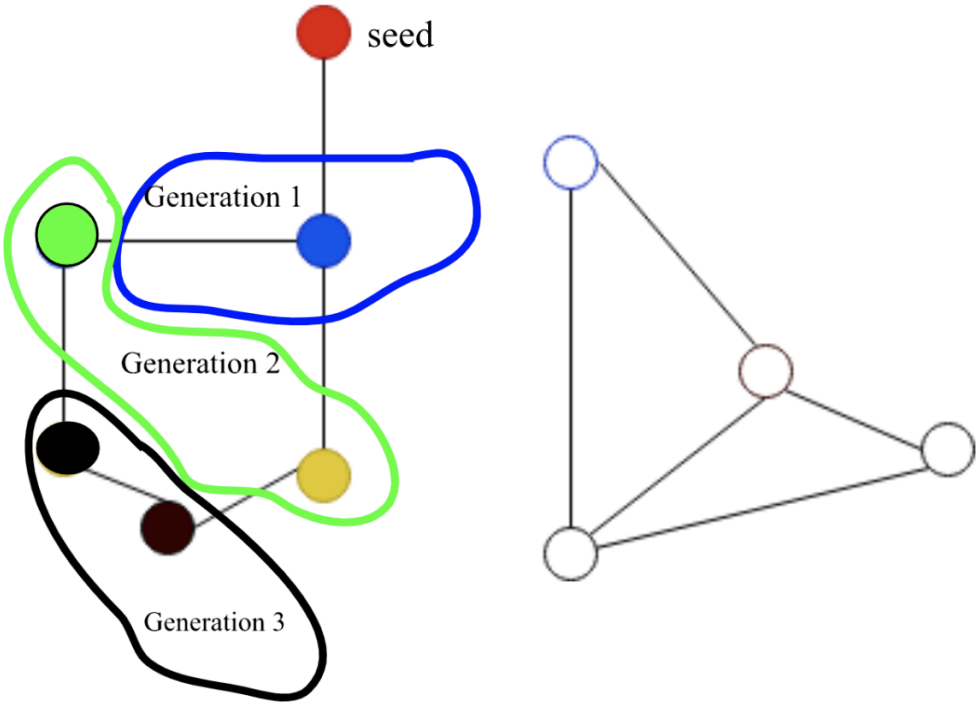
- It will take 3 generations to reach 100% of the nodes in the graph
- It will take 2 generations to reach 60% of the nodes in the graph

- It will take 1 generation to reach 20% of the nodes in the graph



Given the following graph and a seed represented by the vertex colored in red,

- It is not possible to reach 100% of the nodes in the graph from the given seed
- It is not possible to reach 80% of the nodes in the graph from the given seed
- It will take 3 generations to reach 60% of the nodes in the graph
- It will take 2 generations to reach 40% of the nodes in the graph
- It will take 1 generation to reach 20% of the nodes in the graph



Implement and test `generations`. It implements the algorithm described above.

> **Corner cases:**
>
> - You will return -1 if we cannot reach the target threshold.
> - You will return -1 if the seed is outside of the bounds of the nodes in the network
> - You will return -1 if the threshold is negative or greater than 1
> - You will return 0 if the seed is valid but the threshold is 0

Step 5: Degree immunization strategy

The first (immunization) approach that we will explore will use the degree of the nodes in the graph. ***We want to "remove" all the edges incident to the nodes with a specific degree*** and see if it slows the pathogen's spread (fewer spread cycles). Here we will target vertices with high degree.

> **NOTE**
> When we say "remove" all the edges incident to the nodes, we remove the **edges** only, do not remove the node from the network! Thus, when calculating `generationsDegree` and `rNumberDegree`, the total node count should remain constant regardless of edge removals.

> Also, do not *permanently* modify the graph in this step! If you want to remove edges, do so in a COPY of the original graph.

Implement and test the following operations:

- `degree`; it will return the degree of a given node or -1 if the node is not in the network.
- `degreeNodes`; given an integer d passed as a parameter, degreeNodes will return a collection of all the nodes with degree of d in the network. Return an empty collection if there is no node in the network with that degree.
- `generationsDegree`; the method returns the number of generations needed to infect a given percentage (threshold) of the population when all the nodes with a given degree are immunized (have their edges removed) from the network.
- Compute the **R0** (`rNumberDegree`) of this immunization strategy.

> **Corner cases:**
>
> - You will return -1 if we cannot reach the target threshold.
> - You will return -1 if the seed is outside of the bounds of the nodes in the network
> - You will return -1 if the threshold is negative or greater than 1
> - You will return -1 if no node in the graph has the given degree value
> - You will return 0 if seed is removed as part of the immunization

Step 6: Clustering Coefficient immunization strategy

The second immunization approach that we will explore will use the clustering coefficient of the nodes in the graph. We want to "remove" all the edges incident to nodes with a clustering coefficient within a given range and see if it slows the spread of the pathogen (fewer spread levels).

The local clustering coefficient of a vertex (node) in a graph quantifies how close its neighbors are to being a clique—read about how to compute the clustering coefficient of a node here!

Implement and test the following operations:

- `clustCoeff`; given a node n, the method will return its clustering coefficient. Return -1 if the node is not in the network.
- `clustCoeffNodes`; it will return a collection of all the nodes with clustering coefficient between a lower bound and an upper bound, both inclusive. The bounds are passed as parameters to

the method. Return an empty collection if there is no node in the network with that clustering coefficient.

- `generationsCC`; the method returns the number of generations needed to infect a given percentage (threshold) of the population when all the nodes with clustering coefficients within the given range are removed (immunized) from the network.
- Compute the **R0** (`rNumberCC`) of this immunization strategy.

> **Corner cases:**
>
> - You will return -1 if we cannot reach the target threshold.
> - You will return -1 if the seed is outside of the bounds of the nodes in the network
> - You will return -1 if the threshold is negative or greater than 1
> - You will return -1 if no node in the graph with a clustering coefficient within the given range
> - You will return 0 if seed is removed as part of the immunization

NOTE

- The clustering coefficient of a node with degree <=1 is 0
- When comparing clustering coefficient values, allow for a tolerance of ±Δ = 0.01. The static method `inRange` implements this behavior for you.

Step 7: Clustering Coefficient and Degree immunization strategy

The last (immunization) approach that we will explore will combine the clustering coefficient and the degree of the nodes in the graph. Since nodes with low clustering coefficients are less likely to form a clique, and nodes with high degrees are highly connected, we want to see if nodes with high degree and low clustering coefficients are more likely to spread the pathogen faster. We want to "remove" all the edges incident to the nodes with low clustering coefficient and high degree and see if it slows the spread of the pathogen (fewer spread levels).

Implement and test the following operations, keeping in mind that you should be able to accomplish these methods using work that you've already done:

- `highDegLowCCNodes`; it will return a collection of all the nodes with clustering coefficients less than or equal to a given value and with degrees greater or equal to a given degree value. Return an empty collection if there is no node in the network with that degree.

- `generationsHighDegLowCC`; the method returns the number of generations needed to infect a given percentage (threshold) of the population when all the nodes with low clustering coefficient and high degree are removed (immunized) from the network.
- Compute the **R0** - `rNumberDegCC` of this immunization strategy.

> **Corner cases:**
>
> - You will return -1 if we cannot reach the target threshold.
> - You will return -1 if the seed is outside of the bounds of the nodes in the network
> - You will return -1 if the threshold is negative or greater than 1
> - You will return -1 if no node in the graph meets the clustering coefficient and degree requirements
> - You will return 0 if seed is removed as part of the immunization

Step 8: Submit to Gradescope

Do not forget to polish your code. Only submit `InformationSpread.java`, `InformationSpreadTest.java`, README, and your own datasets (only the ones you have created on your own, NOT those provided).

Grading

Your grade will consist of points you earn for submitting the homework on time and for the final product you submit.

The assignment is worth 205 points.

Description	Points
Autograder	135
Testing (correctness & code coverage)	50
Style	15
Readme	5

This assignment was created by Eric Fouh.

References: https://web.stanford.edu/class/earthsys214/notes/R0.html

---

Spring 2023