MCS 320 Project One : Exploring Connectivity in Random Graphs
due Monday 25 February 2019, at 4PM.

We use SageMath in a computational exploration of graphs. A good way to start this project is to download a SageMath notebook from the course web site and execute the statements in that notebook.

0. Unweighted Undirected Graphs

An unweighted undirected graph is defined by vertices and edges. The vertices are also often called the nodes in the graph. Vertices may be connected by edges.

A common definition of a graph is via its adjacency matrix. The adjacency matrix `A` for a unweighted undirected graph is a symmetric matrix of zeros and ones, defined as follows:

1. For every edge between vertex `i` and `j`, the entry `A[i][j]` is one.
2. If there is no edge between vertex `i` and `j`, then `A[i][j]` equals zero.

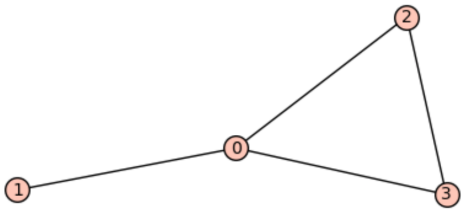
Consider the SageMath code below:

```
A = matrix(4,[[0,1,1,1],[1,0,0,0],[1,0,0,1],[1,0,1,0]])
print A
print 'Is there and edge between 1 and 0? ', A[1][0] == 1
```

which prints the following:

```
[0 1 1 1]
[1 0 0 0]
[1 0 0 1]
[1 0 1 0]
Is there and edge between 1 and 0?  True
```

With the matrix `A` we make a graph, e.g.: as `g = Graph(A)`. To verify, we can retrieve the adjacency matrix of `g` applying the method `adjacency_matrix()` to the variable `g` as `g.adjacency_matrix()`. With the methods `vertices()` and `edges()` we retrieve respectively the vertices and edges of a `Graph` object. Visualizing a graph `g` happens via `g.show()` as shown below:



In a connected graph, there is only one connected component, as there is a path of one or more edges between each pair of vertices. If there is a pair of two vertices for which there exists no path that connects the vertices, then the number of connected components is larger than one. On the example `g`, `g.connected_components()` returns `[[0, 1, 2, 3]]`.

With `g.all_paths(1, 3)` we compute all paths between the vertices `1` and `3`. The shortest path between `1` and `3` is computed with `g.shortest_path(1, 3)`. All shortest paths from vertex `1` are computed by `g.shortest_paths(1)` which returns a dictionary:

```
{0: [1, 0], 1: [1], 2: [1, 0, 2], 3: [1, 0, 3]}
```

1. Random Graphs

Calling the function `random()` returns a number in the interval $[0, 1]$, a number we can interpret as a probability. For reproducibility of results, we can fix the random seed to any number, e.g.: `123456789` with `set_random_seed(123456789)`.

The first important parameter is the probability p that two vertices are connected. With a loaded coin, we can generate a random graph. If the probability p that two vertices are connected is 0.1, then only if `random()` returns a number less than 0.1 will there be a one in the adjacency matrix for any pair of two vertices. The code snippet below generates a sequence of 20 bits, which can define one row in an adjacency matrix of a random graph, where the probability p that two vertices are connected is 0.1.

```
p = 0.1
set_random_seed(123456789)
coinflip = lambda: (1 if random() < p else 0)
[coinflip() for _ in range(20)]
```

The second parameter in our study of random graphs is the number of vertices, which will be the dimension n of the graph. We flip the loaded coin for all entries in the adjacency matrix `A`. To make the matrix A symmetric, for all $i > j$, we set `A`$[i][j]$ to `A`$[j][i]$.

We consider a random graph G with n vertices and probability p of an edge between any two vertices.

Assignment One. Define a function to return a random graph, given the probability p and dimension n. The function must use your UIN as the random seed, so the same graph will be returned for every same value of p and n.

In your answer, give the code for your function and demonstrate the correctness of your function with some well chosen examples.

2. Connectivity

A natural question is then the following. For given n and p, what is the probability that G is connected? In a social network, an edge models a connection between two people. If the dimension of the network grows, then the probability that two people are connected will get smaller.

Assignment Two. Use your function of Assignment One in your experiments to generate random graphs for various probabilities p and dimensions n larger than 20. What is the threshold value $p_{u,n}$ for p such that your generated graphs are connected? The value $p_{u,n}$ depends on your UIN and the dimension n.

What is the evolution of $p_{u,n}$ as n increases? Take sufficiently many large values for n so a trend is noticeable. Organize your experiments with repeatable loops which produce tables of results.

3. Shortest Paths

The second question we are then asking concerns the shortest path between two vertices in a connected graph. In a fully connected graph, the shortest path is one edge, but in a real, sparse graph, many edges may be necessary to reach a vertex from any given vertex.

Assignment Three. In Assignment One, you defined a function to generate your own random graph. In Assignment Two, you applied the function to determine the threshold probability p_t for which the generated graph is connected. The goal of this assignment is to determine the length L of the shortest paths for random graphs of dimension n, generated with probability $p_{u,n}$.

What is the evolution of L as n increases? What is the average value for L, over all pairs of vertices? Organize your experiments with repeatable loops which produce tables of results.

4. The deadline is Monday 25 February 2019, at 4PM.

The solutions to the project will be collected at the beginning of our class meeting on Monday 25 February at 4PM. If you cannot come to class that day, then you must arrange to hand in your solution before the deadline. Otherwise, your solution will be discounted with 10 points if it is turned in on the same day before 11PM, and will no longer be accepted afterwards.

Your solution to the project must be printed on paper. As a backup, send the `.ipynb` file of the project as an attachment in an email to `janv@uic.edu`, before the deadline.

The solution consists of the print out of one single notebook, organized properly to the two assignments. All cells in the notebook should run from top to bottom as a program without errors. Also write proper documentation for the calculations. Apply proper formatting in your notebook so it reads like a technical report.

This project must be solved *individually*. Under no circumstances is it allowed to copy or to collaborate. Regardless of who copied from whom, all caught in the act of plagiarism will be penalized.

If you have questions, comments, or difficulties, feel free to come to my office for help.