Project 3: Giant component of a random graph

Due: 11:59pm, Tuesday, May 19

Collaboration Policy: You can discuss topics related to programming or graphs with your classmates or others. You may also do research on the internet or from books (this is certainly not required). But you must design and write your own algorithms and write your own report. In particular, you should understand how your code works and be able to justify your design choices.

Introduction

In this project you will explore the "giant component" of a random graph. Recall that the vertices of an (undirected) graph G can be partitioned into sets called *connected components*, such that every pair of vertices in each set can be joined by a path consisting of edges of the graph, and no pair of vertices from different sets can be joined by such a path (see the figure below). The number, size, and structure of the connected components are important properties of graphs that arise when modeling networks.

You will investigate the relationship between the connected components (particularly the size of the largest one - the *giant component*) in a random graph and the number of edges in the graph. With few edges, all the connected components are small and there are many of them. As the number of edges increases, eventually it is becomes very likely that almost all vertices in the graph can be joined by a path (with just a few outlying islands of vertices). But between these two extremes there is a transition point when the graph starts to have one large component, and many smaller ones. You will empirically explore this transition point.

Programming tasks [70 pts]

(1) `[20 pts]` Write code that generates a random graph G with n vertices, where each edge appears with probability p. This means that two distinct vertices are connected to each other with an edge with probability p. You can choose how you want to represent the graph G. (Note that this is model is somewhat different than the `rand_graph` function we wrote in lecture, though it has similar properties).

(2) `[5 pts]` Plot visual representations of random graphs for various values of n up to 50, and various values of p. You can use the package `networkx` or some other way of plotting.

(3) `[20 pts]` Write a function that takes in a graph G and returns a list of the connected components. For instance, for the graph in the figure below, the list of the connected components is:

$$[[0, 1, 2, 3, 4, 5], [6, 7, 8, 9], [10]].$$

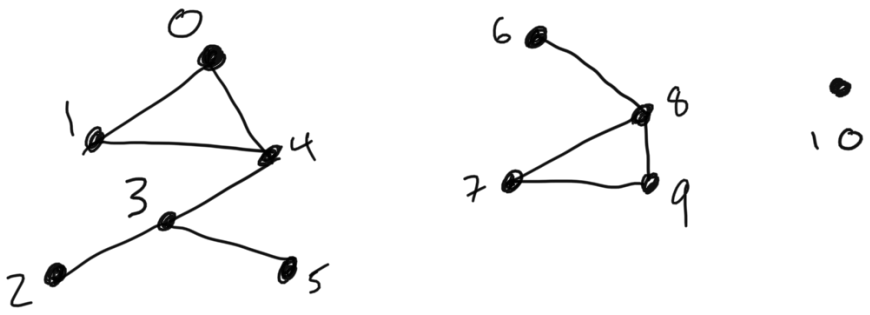

Figure 1: A graph with 3 connected components.

Test out your code on some of the graphs you generated in the previous part.

Hint: In Lecture on Tuesday 5/5 we write code to determine whether a graph is connected. You can modify this a bit to generate a list of connected components.

(4) `[10 pts]` Write a function that takes in a graph and outputs the size of the giant component (measured in number of vertices). Test this for a random graph with $n = 50$ and $p = 1/50$.

(5) `[15 pts]` You will now see what happens to the size of the giant component of random graphs for p near $1/n$. If $p = 1/n$, then the number of edges hitting a vertex v (the degree of v) is around 1. Compute the size of the giant component for random graphs with $n = 10, 50, 100, 200$, and $p = t/n$ where t ranges from 0 to 4 in steps of size 0.25. Do 10 trials for each of the possible (n, p) values and take the average of the size of the giant component over those 10 trials. Generate a plot of your results, with one curve for each of the possible values of n. Make the x-axis the value of pn (which is the average degree of a vertex), and the y-axis the value of the size of giant component divided by n.

From this plot, you should see the emergence of a giant component as p increases from below $1/n$ to above this threshold.

It may be helpful to optimize your code from part (3) so that the above experiments don't take too long (also see Extra Credit (6a) below).

(6) **Extra Credit** `[Up to 8 pts]` You may choose at most *one* of the following to work on for extra credit.

(a) Optimize your code from part (3) so that you can compute the list of connected components efficiently for random graphs with n vertices and $p = 1/n$, for larger n than were studied in part(5). The number of points you get will depend on how efficient your algorithm is.

Hint: You may want to change the way you represent the graph. Since a random graph with n vertices and $p = 1/n$ is relatively sparse, it may be faster to use a representation other than an adjacency matrix. For instance, you could use *adjacency lists*: for each vertex, store a list of the vertices that it neighbors. For reference, on my newish MacBook Pro, with this optimization, I can find connected components for $n = 2000$, $p = 1/2000$ in about 3 seconds. You will receive most of the 8 points if you achieve a similar level of efficiency.

(b) Do experiments and create a plot as in part (5), but this time instead of computing the size of giant component, compute the ratio of the size of the giant component to the size of the second largest component.

This quantity should also experience a transition around $p = 1/n$; below this threshold the giant component isn't much bigger than the second largest component, while above the threshold, the giant component gets much bigger than the second largest component.

(Note that sometimes the whole graph will be connected, so there will be no second largest component. However, when $p > 0$ for our random graphs, this is not very likely, so it doesn't really matter. You can define the ratio to be some large number, like n, if the graph is connected).

Written report [30 pts]

- Describe the algorithm that you used to compute connected components in (3). How do you think the running time would scale with the number of vertices n and the number of edges in the input graph?
- Interpret the results about the size of the giant component you computed in (5). What kind of transition occurs around $p = 1/n$? What do you think would happen if you were able to test random graphs with very large n?
- What are some other questions about properties of random graphs that you could explore in a similar fashion?

What to turn in

You will submit everything on Blackboard. Include the following:

- A `.ipynb` notebook file named `code_firstname_lastname.ipynb` with all your code from the **Programming tasks** section. Make sure to clearly separate the various parts. Also, test this notebook by restarting it and running from the beginning; this should produce no errors.

- A `pdf` file named `report_firstname_lastname.pdf` with your work from the **Written report** section. The exact format of this report is up to you. It can be hand-written (legibly) and then scanned (legibly) into a `pdf` document. Or it can be be made on a computer. It must be in `pdf` form (it is generally easy to convert other forms such as .doc to `pdf`).