CS303–Spring, 2018 Programming Assignment 1 Out: January 30, 2018
Due: Feb. 15, 2018

Homework will be due at the **BEGINNING** *of class on the date due. Your answers should be written legibly if handwritten and your name should be clearly written on your homework assignment. Try to make your answers as clear and concise as possible; style will count in your overall mark. Be sure to read and know the collaboration policy in the course syllabus. Be sure to check the back of the page; problems occasionally show up there too!*

Assigments are expected to be turned in electronically in pdf format (by using latex).

An Erdos-Renyi random graph G(n,p) is generated by two parameters

- n: the number of vertices.
- p: a number in [0,1], which is the probability that each of the n(n-1)/2 edges is independently included in the graph.
- You can check wiki for more details.

Problems

1. **Generating a random graph.** Write a function to generate a random graph given n and p.
    - Input: an integer $n > 1$ and real number p in $[0, 1]$ Output: an undirected graph G. You can use whatever format to store the graph (e.g., adjacency list or matrix). Hint: Initiate G to be an empty graph (no edges). For each $i < j$, generate a random number q between 0 and 1. If $q < p$ then add the edge (i, j) to G (if you are using adjacency list then this involves updating the lists of both i and j). In Python, you can generate a random number in [0,1] using random.random() after importing random.

2. **Computing the size of the largest connected component.** Given a graph G and a threshold t, write a function to test whether G contains a connected component of t vertices or more.
    - Input: a graph G and and a number t
    - Output: 1 if G contains a connected component of t vertices or more; 0 otherwise.
    - Hint: One solution is the following. For each vertex, you can compute the number of vertices in its connected component by BFS or DFS. Then, the function returns 1 if there exists a vertex whose connected component has at least t vertices. There are much faster ways to do this using BFS or DFS.
    - Please test your code thoroughly! A lot of students double-count the nodes, ending up with incorrect component sizes, sometimes even with components larger than the number of nodes in G, which is impossible. To test this, return a list of all the nodes in a connected component (not just its size), and then make sure that no node appears in your list twice, and that the number of nodes in the list equals the size of the component which you computed.

3. **Testing your algorithm on randomly generated graphs.**
    - Let $n = 40$.

- For each c in $[0.2, 3.0]$ with step size 0.2, let $p = c/n$, generate 500 random graphs $G(n, p)$ using your function for step 1.
- Use your function for step 2 to calculate the percentage of graphs (out of the 500 graphs with the same c) whose largest connected component has at least t=30 vertices.
- Output: a graph where the x-axis is c and the y-axis is the percentage. As c increases, more and more graphs should have large components, since they have more and more edges. Hint: The total number of graphs you should generate is 500×15—there are 15 different c's and for each c you should generate 500 graphs. Note that the percentage is calculated for each c.

4. **(Optional) Based on your findings, in a random graph of size 40 about how many others does each node need to have edges to, in order for most of the graph to become one connected component? Investigate the same question for other values of n: how large does c needs to be in order for most of the nodes in the graph to become connected with high probability?**

Other hints

- If your computer is produced after 2011 and it is at least as powerful as a regular laptop (meaning that it is not a netbook, tablet, or smartphone), then part 3 should finish in at most a few minutes.
- To debug your code, try small n first.
- Make sure that some calculations are done as float numbers (e.g., it is safer in Python to use p = c/float(n)). Be careful about the automatic integer conversion. You don't have to generate a figure directly from your code. To receive credit, you just need to show the figure to your TA.