Homework 3 Nathan Bick

1. what (roughly) is the time complexity of :

(a) Finding a word in a paper dictionary if the size of the input is the number n of words in the dictionary.

To find a word in a dictionary, first you scan through the letters until you find the latter corresponding to the first letter in the word of interest.

They key observation is that the dictionary is sorted alphabetically.

once in the section of the appropriate letter it is then necessary to scan through the words to find the correct word.

S-ppose there are n words in the dictionary.

2. Consider the following situations:

(a) You are asked to calculate the closeness centrality of a single node in an undirected network with m edges and n nodes. what algorithm would you use to do this and what would be the time complexity of the operation in terms of m and n?

(b) You are given a road map and told the average driving time along each road segment, then you are asked to find the route from A to B with the shortest average driving time. what algorithm would you use to do this and what would be the time complexity of the calculation?

(c) what algorithm would you use to find all the components in an undirected network, and what would be the time complexity of the operation?

(a) First we recall the definition of closeness centrality, which is given by $C_i = \frac{1}{\ell_i} = \frac{n}{\sum_j d_{ij}}$, where d_{ij} is the shortest distance between nodes i and j.

Intuitively, this means that to calculate the closeness centrality for a single node, we must calculate the shortest distance from that node to all others in the network.

We know from the book that the algorithm to use to calculate the shortest distance is breadth-first algorithm. The time complexity of this algorithm is $O(m+n^2)$, determined by first ~~searching~~ going through neighbors $O(\frac{m}{n})$, passing at each node once, $n \cdot O(\frac{m}{n})$, and repeating for many rounds $O(n + rn + m) = O(m+n^2)$.

(b) This roadmap is an example of a network with varying edge lengths. The shortest path in this case is not the path with the fewest number of edges, necessarily. We therefore must use Dijkstra's algorithm, rather than breadth-first.

In a simple implementation, we search thru all nodes in each round to find the one with shortest distance, taking $O(n)$. Then, calculate distance to each neighbor of the node we find, $O(\frac{m}{n})$, giving $O(\frac{m}{n}+n)$. But repeat for n rounds, yielding $O(n^2+m)$.

If a heap is used, the text explains that the above can be optimized to $O((m+n)\log n)$ time complexity.

(c) To find all components in an undirected network, one could use an algorithm where one does breadth-first search to find if there are paths between all pairs of nodes. We know this takes $O((m+n)n)$, or $O(n^2)$ for sparse networks

3. What is the time complexity, as a function of the number n of nodes and m of edges, of the following network operations if the network is stored in an adjacency list?

(a) Calculating the mean degree.

(b) Calculating the median degree.

(c) Calculating the air-travel route between two airports that has the shortest total flying time, assuming the flying time of each individual flight is known.

(d) Calculating the minimum number of routers that would have to fail to disconnect two given nodes on the Internet.

(a) In an adjacency list, we would have to first scan through each row of the list and count the number of ~~nodes~~ edges indicated for each node, giving the degree. Then we would need to sum these degrees and divide by n to get the mean degree

Listing the neighbors like above takes $O(\frac{m}{n})$ for each node, so $O(m)$ for all nodes. I believe the mean calculation would be $O(m)$.

(b) To calculate the median degree, the first step would be to calculate the degree for each node. Given this information, it would then be required to sort the nodes by their degree. At that point, the final operation would be to calculate the value in the "middle" of the set of nodes' degree values.

To sort, one would do $O(\frac{m}{n}) \cdot O(n) = O(m)$ to get the degrees. Then to move things around that could take $O(n)$ at worst, so it should be $O(nm)$.

(c) To calculate the air-travel route with the shortest flying time, we would be in the case of a network with varying edge lengths. This would suggest us to use Dijkstra's algorithm, which has time complexity $O((m+n)\log n)$ in its efficient implementation, as we see in the text.

(d) As explained in the text the problem of finding the minimum number of edges that need to be removed to disconnect two nodes, which this router question is an example of, is equivalent to the independent path or max flow problems. Therefore, ~~bath~~ the book explains that the min cut set algorithm is the max flow algorithm. The most common example is the Ford-Fulkerson or augmenting path algorithm, which has time complexity $O((m+n)m/n)$.

4. For an undirected network of n nodes and m edges stored in adjacency list format show that:

(a) It takes time $O(n(m+n))$ to find the diameter of the network.

(b) It takes time $O(\langle k \rangle)$ on average to list the neighbors of a node, where $\langle k \rangle$ is the average degree in the network, but time $O(\langle k^2 \rangle)$ to list the second neighbors.

(a) We recall the definition of the diameter of the network as the longest shortest path between any two nodes. Therefore, we would consider the breadth-first algorithm, which takes $O(m+n)$ time complexity to find shortest distance to all nodes from a single source node. This must be repeated for all n nodes as source node, yielding time complexity $O(n(m+n))$.

(b) In the adjacency list, each node on average has k items in its row of data indicating its edges. To list the node's neighbors requires scanning across the items in the row of data, which on average is $\langle k \rangle$ items, so the time complexity is $O(\langle k \rangle)$.
To list the second neighbors, one first repeats the above, which is $O(\langle k \rangle)$. The second step is to jump to the row of data corresponding to each neighbor ~~and~~ in turn, and each time listing its neighbors, which includes $\langle k \rangle$ items on average.
Therefore, the time complexity is $O(\langle k^2 \rangle)$.

5. For ~~a~~ a ~~un~~directed network in which in- and out-degrees are uncorrelated, show that it takes time $O(m^2/n)$ to calculate the reciprocity of the network. Why is the restriction to uncorrelated degrees necessary? What could happen if they are correlated?

First we recall the definition of reciprocity r, given by $r = \frac{1}{m} \sum_{ij} A_{ij} A_{ji} = \frac{1}{m} \operatorname{Tr} A^2$.

If we consider that the network in/out degrees are uncorrelated, we would need to check if a node is connected, which takes $O(\frac{m}{n})$ as explained in the book, and this most be checked for all edges, so $O(m)$.

Therefore, $O(\frac{m^2}{n})$ is the time ~~that~~ complexity.

We now consider the correlation by considering the extreme cases of perfect correlation and perfect negative correlation.

If perfect correlation, for all networks of size n,

$r = \frac{1}{m} \cdot n = \frac{n}{m}$ This doesn't allow for comparison at all. This is by considering $\operatorname{Tr} A^2$.

If negative correlation, similarly than $r = \frac{1}{m} \cdot 0 = 0$

Therefore, we see that ~~it~~ in the limit, for the correlated network, it is not as useful to consider reciprocity.

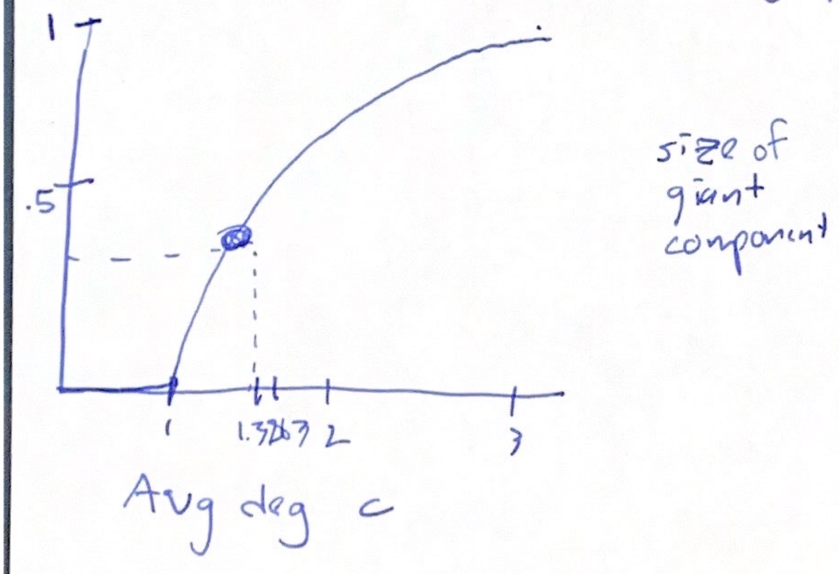
6. Let's create an analytic prediction for the giant component of $G(n,p)$.

We denote u as the average fraction of nodes that do not belong in the giant component.

We have $u = (1-p+pu)^{n-1}$. Following the algebra of the text we arrive at $S = 1 - e^{-cS}$, which is the fraction of nodes in the largest component.

This doesn't have a simple closed-form solution for S, but allows for some graphical investigation, like in Figure 11.2.

In Fig 11.2 (b), we see a graph similar to below.



In our case $c = 1.3863$, so we should expect a size lower but close to 0.5.

This is close to what I get in my script, which is 0.48, approximately.