CS 330, Fall 2019, Homework 3, Due Wednesday, September 25, 2019

Homework Guidelines

Please make sure you read the collaboration policy before you start working on your homework. Refer to the general information handout for the homework policy and additional options.

Collaboration policy Collaboration on homework problems, with the exception of programming assignments and reading quizzes, is permitted, but not encouraged. If you choose to collaborate on some problems, you are allowed to discuss each problem with at most 5 other students currently enrolled in the class. Before working with others on a problem, you should think about it yourself for at least 45 minutes. Finding answers to problems on the Web or from other outside sources (these include anyone not enrolled in the class) is strictly forbidden.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." It is a violation of this policy to submit a problem solution that you cannot orally explain to an instructor or TA.

Solution guidelines For problems that require you to provide an algorithm, you must give the following:

1. a precise description of the algorithm in English and, if helpful, pseudocode,
2. a proof of correctness,
3. an analysis of running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class.

You should be as clear and concise as possible in your write-up of solutions. Understandability of your answer is as desirable as correctness, because communication of technical material is an important skill. A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand. Points might be subtracted for illegible handwriting and for solutions that are too long. Incorrect solutions will get from 0 to 30% of the grade, depending on how far they are from a working solution. Correct solutions with possibly minor flaws will get 70 to 100%, depending on the flaws and clarity of the write up.

Problems to be handed in (see solutions guidelines above for the format)

Problem 1 *(10 pts.) Rumor propagation*

In lecture we discussed Milgram's 1967 experiment; he picked 300 people at random in Nebraska and asked them to send a letter to a stockbroker in Boston, by way of relaying the letter through a chain of people. The rule is that every person has to know the next person they are sending the

letter to on a first name basis. He found that on average each letter went through the hands of 6.4 people before reaching the stockbroker. This is where the expression "six degrees of separation" comes from.

When you tell your friend Dirk about this experiment he says he is not surprised. Dirk says that often, when he tells something to a friend, a couple of days later he hears back the same information from someone else! You decide to test whether the six degrees of separation principle can also be applied to oneself.

Let's imagine that "friendships" on Facebook are a good representation of Milgram's rule for being on first-name terms with somebody. Assume you are given access to all of the friendship links on Facebook as a graph (where nodes are accounts, and links are "friends"). Design an algorithm to determine if there is a chain of at most 7 friends (because the average number in Milgram's experiment was 6.4) such that Dirk is friends with both the first and the last person. You may assume that the graph is undirected.

For full credit your algorithm should run in time $O(m+n)$ (where n and m are the number of nodes and edges, respectively).

Hint: Modify/use the BFS algorithm from class.

Problem 2 *(10 pts.) Strongly connected components*

Let $G(V, E)$ be a simple *directed* graph with n nodes and m edges. Devise an algorithm that takes G as input and outputs the *strongly* connected components of G. The output should be a set of lists of nodes (one per component), where each list contains the nodes of a single component. For full credit your algorithm must run in time $\Theta(n+m)$ regardless of the number of strong components. For partial credit you can submit an algorithm with running time that is a function of the number of strongly connected components.

Hint: In class, we saw an algorithm that finds a *single* strongly connected component in time $O(m+n)$. Use that algorithm as a starting point.