CS41 Homework 10

This homework is due at 11:59PM on Sunday, November 18. Write your solution using LaTeX. Submit this homework using **github** as a **.tex** file. This is a **partnered homework**. You should primarily be discussing problems with your homework partner.

It's ok to discuss approaches at a high level with others. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you've done with a lab partner *while in lab*. In this case, note (in your **README** file) who you've worked with and what parts were solved during lab.

1. **Longest Common Subsequence.** A *subsequence* in a string is a sequence of letters that appears in the same order, but not necessarily contiguously. For example, "alg", "arhm", "goth", and "loihm" are all subsequences of "algorithms", but "xyz", "algog", "gla", and "rihtms" are not subsequences.

   A *common subsequece* between two strings a, b is a string c that is a subsequence of both a and b. For many applications including in computational biology, we would like a way of finding the largest common subsequence between two strings.

   Design an algorithm that takes as input two n-character strings a, b and returns the size of the largest common subsequence between them.

2. **Flow variant.** In the standard flow problem, we get an input $G = (V, E)$ a directed graph and edge capacities $c_e \geq 0$ limiting how much flow can pass along an edge. Consider the following two variants of the maximum flow problem.

   (a) It might be that each junction where water pipes meet is limited in how much water it can handle (no matter how much the pipes can carry). In this case, we want to add *vertex* capacities to our problem. The input is a directed G (with source s and sink $t \in V$), edge capacities $c_e \geq 0$, and vertex capacities $c_v \geq 0$ describing the upper limit of flow which can pass through that vertex. Give a polynomial-time algorithm to find the maximum $s \rightsquigarrow t$ flow in a network with both edge and vertex capacities.

   (b) It might be that there are multiple sources and multiple sinks in our flow network. In this case, the input is a directed G, a list of sources $\{s_1, \ldots, s_x\} \subset V$, a list of sinks $\{t_1, \ldots, t_y\} \subset V$, and edge capacities $c_e \geq 0$.

   Give a polynomial-time algorithm to find the maximum flow in a network with multiple sources and multiple sinks.

3. **Hospitals coping with natural disaster.** (K&T 7.9)

   The same hospitals from earlier in the semester have now hired all the doctors they need. There is a widespread natural disaster (like the fires in California!), and a lot of people across an entire region need to be rushed to emergency medical care. Each person should be brought to a hospital no more than 50 miles away from their current location. Additionally, we want to make sure that no single hospital is overloaded, so we want to spread the patients across the available hospitals. There are n people who need medical care and h hospitals; we want to find a way to coordinate emergency medical evacuations so that each hospital ends up with at most $\lceil n/h \rceil$ patients in emergency care. (Also, obviously: every patient should end up at a hospital!)

   Give a polynomial-time algorithm that takes the given information about patients' locations and hospitals and determines whether this is possible. If it is possible, your algorithm should also output an assignment of patients to hospitals ensuring that every patient gets to a nearby hospital and that no hospital is overloaded.

   Prove that your algorithm is correct.

4. **(extra challenge)** Improvements for counting inversions?

   In class we saw a $O(n \log(n))$ time divide-and-conquer algorithm for the problem of counting inversions. Can we do better?

   - If your answer is YES, design and analyze an algorithm that works for Counting Inversion and uses *less* than $O(n \log n)$ runtime.
   - If your answer is NO, give an $\Omega(n \log n)$ lower bound on the number of timesteps taken by any algorithm that counts inversions. Note: it might be easier to prove this lower bound for comparison-basd algorithms.