CSE 417
Algorithms & Computational Complexity
Assignment #2
Due: Friday, 1/21/22

Review the instructions in HW #1. Seriously! I want to especially emphasize that each problem should be on a page (or pages) *by itself*, turned in just to the corresponding dropbox.

1. (FYI, this is the *corrected* version of the problem I botched last week. Your partial progress on it last week may be relevant, but please re-read it carefully to be sure you aren't carrying forward misconceptions. The old version is included, if it is helpful for you to understand what changed, but I recommend, if possible, that you ignore it, and approach with fresh eyes.)

   The Team USA Winter Olympic 5-Person Snowball Throwing Trials are approaching. There are n coaches available; each will coach one 5-Person Snowball Throwing Team ("5PSTT" for short). $5n$ athletes are eagerly competing for these highly coveted 5PSTT slots. After practices, trials and training, each athlete ranks the n coaches, and each coach ranks all $5n$ athletes.

   This is the old, faulty text. Ignore it if possible: A "stable team assignment" assigns 5 athletes to each coach so that no coach c is assigned and athlete a who would prefer a different coach c' and simultaneously c' is assigned an athlete a' who prefers coach c.

   New Text: A "team assignment" partitions the athletes into n groups of 5 and matches each group to a coach. Such an assignment is "stable" if it contains no "instabilities." An instability in such an assignment is a pair of athletes a_1 and a_2 and a pair of coaches c_1 and c_2 such that a_i is on c_i's team ($i = 1, 2$), but coach c_1 would prefer to have a_2 and a_2 would rather be on c_1's team.

   This is similar, but not identical, to the Stable Matching Problem (SMP) in the text. Explain how you could use, *without modification*, a subroutine that solves SMP to solve the Stable 5PSTT Problem (S5PSTTP). I.e., explain how to transform an S5PSTTP instance x into an SMP instance $f(x)$ in such a way that any stable solution to the SMP instance $f(x)$ can be easily converted into a stable solution to the S5PSTTP instance x. I'm looking for about 3 clear, English language, paragraphs explaining (1) how to convert x to $f(x)$ (2) how to recover a solution to the S5PSTTP instance x from a Stable Matching found in $f(x)$, and (3) explaining why that result is actually a "stable team assignment". I expect that your solution would require $O(n^2)$ time (which is linear in the size of the data that describes a problem instance, namely a pair of $5n \times n$ preference matrices), *excluding* the time taken by the SMP subroutine; *briefly* justify this (if true), or explain why your method may be slower.

2. The spanning tree produced by Breadth First Search in an undirected graph is a just set of edges. (It is not the drawing of those nodes and edges that we typically use to illustrate the algorithm.) The algorithm does not specify the order in which edges incident to a visited node are added to the queue, and different orders may result in different spanning trees (all equally valid). For BFS on the graph used as an example in lecture (approximately slides 37-40), always starting from node 1, how many different spanning trees are possible and briefly describe how each arises and what changes compared to the picture on slide 40.

3. The gist of the problem above applies equally well to Depth First Search—different, equally valid, DFS spanning trees may arise depending on the order in which you explore the edges incident to a visited node. Consider the graph used as an example in lecture (approx slides 72-110), but for simplicity *remove* the two edges $\{A, J\}$ and $\{B, J\}$. The examples shown on those slides remains a valid DFS starting from vertex A of this smaller graph, but obviously without those two edges. Always starting DFS from node A, how many different spanning trees are possible and briefly describe how each arises and what changes compared to the picture on slide 110.

4. Give an algorithm that finds a simple cycle in an undirected graph, if it has one, or reports that it does not. A "simple cycle" is a list of vertices, with no vertex repeated, such that there is an edge connecting each successive pair of vertices, and one connecting the last to the first. A graph may have many cycles; only report one.

   NOTE: Read the course FAQ page to understand what the code words "give an algorithm" mean. Seriously!

5. Let $G = (V, E)$ be a connected, undirected graph, and let s be a fixed vertex in G. Let T_B be the spanning tree of G found by a bread first search starting from s, and similarly T_D the spanning tree found by depth first search,

also starting at s. (As in problem 1, these trees are just sets of edges; the order in which they were traversed by the respective algorithms is irrelevant.) Using facts presented in lecture and/or the text, prove that $T_B = T_D$ if and only if $T_B = T_D = E$, i.e., G is itself a tree.

6. Mikileaks has posted n (thousands of) blurry images purportedly stolen from a secrect government database of UFO sitings. You believe each potentially belongs to one of two different groups, perhaps Romulans vs Dragons (R and D for simplicity), but it's hard to directly label any one image in isolation.

   Instead, you study each pair i and j side-by-side; label the pair (i, j) either "same" (meaning you believe them both to come from the same group) or "different" (they represent different groups). As a third alternative, you may offer no opinion on a given pair; call the pair *ambiguous*.

   Given the collection of n images, as well as a collection of m judgments (either "same" or "different" for the non-ambiguous pairs), you'd like to know if this data is consistent with the idea that each image is from one of only two groups. More concretely, we'll declare the m judgments to be *consistent* if it is possible to label each image either R or D in such a way that for each pair (i, j) labeled "same," it is the case that i and j have the same label; and for each pair (i, j) labeled "different," it is the case that i and j have opposite labels. For example, if 1 and 2 are "same", and 1 and 3 are "same", but 2 and 3 are "different", then there is no consistent labeling, whereas if 2 and 3 are "same", then there is a consistent labeling.

   Give an algorithm with running time $O(m + n)$ that determines whether the m judgments are consistent.