Practice Homework # 1

The purpose of this first practice homework is to help you review useful background for the course. If you already have had a good algorithms class and remember what you learned, you need not spend much time on this. But if you are rusty or have a background in a different field, then please take advantage of the extra time you have now, early in the semester, to get comfortable with the material covered in these questions.

These questions don’t cover NP-completeness but, as noted in the slides from the first class, I also encourage you to get up to speed on this topic soon, e.g., by reading Chapter 8 of Kleinberg and Tardos’ text, at least through Section 8.4. I also really like Chapters 3 and 4 of Mertens and Moore’s Nature of Computation text.

Practice homeworks won’t be graded, but give you a chance to explore the material covered in class more deeply, identify areas where you are confident in the material versus where you need more background or practice, and help you be fully prepared for the actual homeworks and final exam.

It’s great to discuss these problems with others in the class, and we may touch on them also during class time. But I recommend that you first think about them on your own, to see how far you can get. Please also come to talk with me during office hours if you need more guidance.

Finally, I won’t post solutions to these myself, but if some of you would like to volunteer to share your solutions, that would be fantastic. More than one solution to some problems can be helpful.

1. Asymptotic notation review. Problems 4, 5 of Chapter 2 of Kleinberg and Tardos, slightly adapted.

(a) Take the following list of functions and arrange them in ascending order of growth rate. That is, if function \( g_j(n) \) immediately follows function \( g_i(n) \) in your list, then it should be the case that \( g_i(n) = O(g_j(n)) \).

\[
\begin{align*}
g_1(n) &= 2^{\sqrt{\log n}} \\
g_2(n) &= 2^n \\
g_3(n) &= n\log n^3 \\
g_4(n) &= n^{\log n} \\
g_5(n) &= 2^{2^n} \\
g_6(n) &= 2^{n^2} \\
g_7(n) &= n^2 \\
g_8(n) &= 3^n
\end{align*}
\]

(b) Is it the case that for every consecutive pair \( g_i(n), g_j(n) \) in your list, it is also the case that \( g_i(n) = o(g_j(n)) \)? Why or why not?

(c) Suppose that \( f \) and \( g \) are functions such that \( f(n) = O(g(n)) \). For each of the following statements, decide whether you think that it is true or false. If true, explain why and if false, give a counterexample.

(i) \( \log_2 f(n) = O(\log_2 g(n)) \).
(ii) \( 2^f(n) = O(2^g(n)) \).
(iii) \( f(n)^2 = O(g(n)^2) \).
2. **Graph algorithms.** Adapted from Moore and Mertens.

A connected, undirected graph $G$ has an *Eulerian cycle* if it has a cycle that crosses each edge of $G$ exactly once. Euler claimed that a connected undirected graph contains an Eulerian cycle if and only if every vertex has even degree. If exactly two vertices have odd degree, the graph contains an Eulerian path, i.e., a path that crosses each edge exactly once, but not an Eulerian cycle.

(a) Prove Euler’s claim. (Suggestion: First show that if every vertex has even degree, the graph can be covered with a set of cycles such that every edge appears exactly once. Then see how to combine the cycles into a single cycle.)

(b) Find a simple algorithm that constructs an Eulerian path. (Suggestion: If removing an edge will disconnect the graph, we call that edge a bridge. Now consider the following simple rule, known as Fleury’s algorithm: at each step, consider the graph $G'$ formed by the edges you have not yet crossed, and only cross a bridge of $G'$ if you have to. Show that if a connected graph has two vertices of odd degree and we start at one of them, this algorithm will produce an Eulerian path, and that if all vertices have even degree, it will produce an Eulerian cycle no matter where you start.

3. **Stable matchings (SMs).** For all parts of this problem, to keep things simpler you need concern yourself only with instances of SM where all professors and students are acceptable to each other, each professor seeks one student as a TA, and each student seeks one professor to work with.

(a) We showed in class that the GS algorithm takes at most $nm$ iterations, on an instance with $n$ professors and $m$ students. Find an example with $n$ professors and $m$ students where the number of iterations is as close to $nm$ as possible.

(b) Here is a simple algorithm for SM: Enumerate all assignments (i.e., subsets of professor-student pairs), discard those that are not matchings, then discard those matchings with blocking pairs or unacceptable pairs. Any remaining matching is stable.

Provide a rough bound the running time (number of steps) of this algorithm. Is this a polynomial time algorithm?

(c) Consider the SM problem with ties; that is, professors and/or students may include ties in their preference rankings. Assume that each professor seeks one student TA and each student seeks one professor.

Let $M$ be a matching. A pair $(p, s)$ is *super-blocking* if $(p, s)$ is not in $M$ and both of the following are true.

(i) Either $s$ is unassigned, or $s$ prefers $p$ to $M(s)$, or $s$ is indifferent between $p$ and $M(s)$.

(ii) Either $p$ is unassigned, or $p$ prefers $s$ to $M(p)$, or $p$ is indifferent between $s$ and $M(p)$.

$M$ is *super-stable* if $M$ has no super-blocking pair.

Generalize the Gale-Shapley (GS) algorithm so that it either finds a super-stable matching for the SM problem with ties, or determines that no such matching exists.