COMPSCI 311: Introduction to Algorithms **Spring 2020**

Homework 1

Released 1/26/2020 Due 2/10/2020 11:59pm in Gradescope

Instructions. You may work in groups, but you must write solutions yourself. List collaborators on your submission. Also list any sources of help (including online sources) other than the textbook and course staff.

If you are asked to design an algorithm, please provide: (a) the pseudocode or precise description in words of the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.

Submissions. Please submit a PDF file. You may submit a scanned handwritten document, but a typed submission is preferred. Please assign pages to questions in Gradescope.

1. **(30+5 points) Variations on Stable Matching**

   (a) (10 points) Consider a variant of stable matching in which at every point, either a *free* college *or* a *free* student can propose. As in the Gale-Shapley algorithm, proposals are done going down in the preference list, so that a proposer cannot repeat a proposal to the same partner. Show that this algorithm always terminates with a perfect matching, but not necessarily a stable one.[1]

   (b) (5 points extra credit) Consider colleges A B, C and students 1, 2, 3, with preference lists:
   1: A B C A: 2 3 1
   2: B C A B: 3 1 2
   3: C A B C: 1 2 3
   How many of the six perfect matchings are stable? How many stable matchings can be obtained by a version of part (a) where colleges and students alternate proposing (any party can start)?

   (c) (10 points) Now allow proposals also from matched parties. That is, on any turn, any college or student may propose to the next partner on its preference list if this is better than the current match (if any), and if the proposal is accepted, both the proposer's and the acceptor's former partner (if any) become free. Does the algorithm always terminate? Will it always produce a perfect matching? Will it always produce a stable matching? (Hint: a college c might not get a student s when s already has a better match, but be offered by s later if s becomes unmatched. Look at $n = 3$ first).

   (d) (10 points) Consider now an algorithm that starts with an arbitrary perfect matching of colleges to students. As long as the matching is not stable, choose an instability (c, s) and eliminate it, by matching c with s, and the former partners of c and s with one another. Show that this algorithm does not always terminate with a stable matching. (Hint: an example with $n = 3$ suffices).

2. **(15 points) Big-O.** For each function $f(n)$ below, find (1) the smallest *integer* constant H such that $f(n) = O(n^H)$, and (2) the largest *positive real* constant L such that $f(n) = \Omega(n^L)$. Otherwise, state that H or L do not exist. All logarithms have base 2. Your answer should consist of: (1) the correct value of H, (2) a proof that $f(n)$ is $O(n^H)$, (3) the correct value of L, (4) a proof that $f(n)$ is $\Omega(n^L)$.

   (a) $f(n) = \frac{n \log n}{\log(\log n)}$

   (b) $f(n) = \sum_{k=1}^{n} \sqrt{k} \cdot \sqrt{n-k}$

   (c) $f(n) = \sum_{k=1}^{n} k \cdot 2^{-k}$

[1] We are no longer confident that the procedure with only free parties proposing always yields a perfect matching. We have also failed to find an example where it doesn't. You will get full credit for exhibiting an example where this procedure fails to find a stable matching. There may be extra credit available if a student or students resolve the question of whether the matching from the procedure is always perfect.

(d) $f(n) = \sum_{k=1}^{n} k \log k$

(e) $f(n) = \sum_{k=1}^{n} \frac{n-k}{k}$

3. **(15 points) Asymptotics.** Let $n = k^2$ for some positive integer k. A Sudoku solution is an $n \times n$ array of numbers, each in the set $\{1, ..., n\}$, such that the same number does not occur (1) twice in a row, (2) twice in a column, and (3) twice in a $k \times k$ square "box", where the whole array is divided into n such boxes. Given an array as input, we want to determine whether it is a Sudoku solution.

   (a) (5 points) Give an $\Theta(n^3)$ time algorithm to solve this problem.

   (b) (5 points) Give an $\Theta(n^2)$ time algorithm to solve this problem.

   (c) (5 points) Give an $\Omega(n^2)$ lower bound for this problem, by showing that any algorithm that makes fewer than n^2 queries to entries of the input array cannot be correct for all possible inputs.

4. **(20 points) Paths of Particular Lengths** Let G be an undirected graph, and s and t be arbitrary vertices of G. We are interested in the possible lengths of (non necessarily) simple paths from s to t.

   (a) (10 points). Design an algorithm that determines whether there is a path from s to t with an odd number of edges. Do the same for an even number of edges. Argue that your algorithm is correct and analyze its running time.

   (b) (10 points). Design an algorithm that determines whether there is a path from s to t whose length is divisible by 3. Argue that your algorithm is correct and analyze its running time. (Hint: It might be useful to also consider paths whose length is 1 or 2 modulo 3).

5. **(20 points) (Non)overlapping Intervals**. We must schedule n jobs, each with a starting time s_i and a finishing time f_i. We have a set of constraints of the following form, each about a pair of jobs:

   - job i finishes before job j starts
   - jobs i and j at least partially overlap.

   We want to assign each of the n starting and finishing times a *distinct* integer value from 1 to $2n$, satisfying these constraints. Design an algorithm that produces such an assignment or reports that this is not possible. Argue that your algorithm is correct and analyze its running time.

6. **(0 points).** How long did it take you to complete this assignment?