CS 124 Homework 2: Spring 2021

Your name:

Collaborators:

No. of late days used on previous psets:
No. of late days used after including this pset:

Homework is due Wednesday 2021-02-17 at 11:59pm ET. You are allowed up to **twelve** (college)/**forty** (extension school) late days through the semester, but the number of late days you take on each assignment must be a nonnegative integer at most **two** (college)/**four** (extension school).

Try to make your answers as clear and concise as possible; style will count in your grades. Be sure to read and know the collaboration policy in the course syllabus. Assignments must be submitted in pdf format on Gradescope. If you do assignments by hand, you will need to scan in your results to turn them in.

For all homework problems where you are asked to design or give an algorithm, you must prove the correctness of your algorithm and prove the best upper bound that you can give for the running time. Generally better running times will get better credit; generally exponential time algorithms (unless specifically asked for) will receive no or little credit. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail, but keep in mind pseudocode does NOT substitute for an explanation. Answers that consist solely of pseudocode will receive little or not credit. Again, try to make your answers clear and concise.

1. **(10 points)** We saw in lecture that we can find a topological sort of a directed acyclic graph by running DFS and ordering according to the postorder time (that is, we add a vertex to the sorted list *after* we visit its out-neighbors). Suppose we try to build a topological sort by ordering according to the preorder, and not the postorder, time. Give a counterexample to show this doesn't work, and explain why it's a counterexample.

2. **(15 points)** News from Cambridge, for those of you far away: every night, snow falls and covers all the sidewalks. Every morning, the city's lone snow shoveler, Pat, is tasked with clearing all the sidewalks of snow. Proper snow-shoveling technique requires that sidewalks on opposite sides of the same street be shoveled in opposite directions. (Every street has sidewalks on both sides.) Give an algorithm to find a snow-shoveling path for Pat that doesn't require any more walking than necessary—at most once per sidewalk. (If you have to assume anything about the layout of the city of Cambridge, make it clear!) (Your algorithm should work for any city, not just the Cambridge in which Harvard is.)

3. **(15 points)** The *risk-free currency exchange problem* offers a risk-free way to make money. Suppose we have currencies $c_1, \ldots, c_n$. (For example, c_1 might be dollars, c_2 rubles, c_3 yen, etc.) For various pairs of distinct currencies c_i and c_j (but not necessarily every pair!) there is an exchange rate $r_{i,j}$ such that you can exchange one unit of c_i for $r_{i,j}$ units of c_j. (Note that even if there is an exchange rate $r_{i,j}$, so it is possible to turn currency i into currency j by an exchange, the reverse might not be true— that is, there might not be an exchange

rate $r_{j,i}$.) Now if, because of exchange rate strangeness, $r_{i,j} \cdot r_{j,i} > 1$, then you can make money simply by trading units of currency i into units of currency j and back again. (At least, if there are no exchange costs.) This almost never happens, but occasionally (because the updates for exchange rates do not happen quickly enough) for very short periods of time exchange traders can find a sequence of trades that can make risk-free money. That is, if there is a sequence of currencies $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdots \cdot r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$, then trading one unit of c_{i_1} into c_{i_2} and trading that into c_{i_3} and so on back to c_{i_1} will yield a profit.

Design an efficient algorithm to detect if a risk-free currency exchange exists. (You need not actually find it.)

4. **(20 points)** Give an algorithm to find the lengths of all shortest paths from a given vertex in a directed graph $G = (V, E)$ where all edge weights are integers between 0 and m, inclusive. Your algorithm should work in time $O(|E| + |V|m)$. (Hint: Modify Dijkstra's algorithm.)

5. **(15 points)** Design an efficient algorithm to find the *longest* path in a directed acyclic graph whose edges have real-number weights. (Partial credit will be given for a solution where each edge has weight 1; full credit for solutions that handle general real-valued weights on the edges, including *negative* values.)

6. **(15 points)** Suppose that you are given a directed graph $G = (V, E)$ along with weights on the edges (you can assume that they are all positive). You are also given a vertex s and a tree T connecting the graph G that is claimed to be the tree of shortest paths from s that you would get using Dijkstra's algorithm. Can you check that T is correct in linear time?

7. **(0 points, optional)**[1] This exercise is based on the 2SAT problem. The input to 2SAT is a logical expression of a specific form: it is the conjunction (AND) of a set of clauses, where each clause is the disjunction (OR) of two literals. (A literal is either a Boolean variable or the negation of a Boolean variable.) For example, the following expression is an instance of 2SAT:

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1}).$$

A solution to an instance of a 2SAT formula is an assignment of the variables to the values T (true) and F (false) so that all the clauses are satisfied– that is, there is at least one true literal in each clause. For example, the assingment $x_1 = T, x_2 = F, x_3 = F, x_4 = T$ satisfies the 2SAT formula above.

Derive an algorithm that either finds a solution to a 2SAT formula, or returns that no solution exists. Carefully give a complete description of the entire algorithm and the running time.

(Hint: Reduce to an appropriate problem. It may help to consider the following directed graph, given a formula I in 2SAT: the nodes of the graph are all the variables appearing in I, and their negations. For each clause $(\alpha \vee \beta)$ in I, we add a directed edge from $\overline{\alpha}$ to β and a second directed edge from $\overline{\beta}$ to α. How can this be interpreted?)

8. **(0 points, optional)**[2] Give a complete proof that $\log(n!)$ is $\Theta(n \log n)$. Hint: you should look for ways to bound $(n!)$. Fairly loose bounds will suffice.

[1] We won't use this question for grades. Try it if you're interested. It may be used for recommendations/TF hiring.
[2] We won't use this question for grades. Try it if you're interested. It may be used for recommendations/TF hiring.