Analysis of Algorithms I: Final Exam, Monday May 8, 2017

This exam ends at 10:10pm. Use the two exam books provided. Write your solutions in one exam book and use the other as scratch paper. Print your name and uni on the cover page. The exam contains five problems. The more difficult items are marked with $*$. Make sure you understand what the question is asking. Do not spend too much time on any single problem. Read them all through first and attack them in the order that allows you to make the most progress. The exam is closed book and closed notes. You may bring ten crib sheets (both sides). You can use any algorithm that we covered in class by simply referring to it and specifying the input. You do not need to repeat the algorithms. Be clear, precise and succinct in your answers. You will be graded not only on the correctness of your answers, but also on the clarity with which you express them. Work on your own! Plagiarism and other anti-intellectual behavior will be dealt with severely. This includes the possibility of failing the course or being expelled from the university.

Problem 1: True or False [24 points, 6 points per question] For each of the **claims** below, answer True or False. No justification is needed. For each question, you will receive 6 points for a correct answer, 0 point for an incorrect answer, and 2 points if you leave it blank.

1. **Claim:** Representing a graph with n vertices in the adjacency matrix representation always takes $\Theta(n^2)$ space no matter how many edges the graph has.

2. Let T denote an undirected tree with n vertices and root r. Let v denote a leaf vertex of T. **Claim:** If T is a complete binary tree, then $\mathsf{DFS}(T, r)$ *always* discovers v in $O(\log n)$ steps.

3. Consider a connected undirected graph $G = (V, E)$ with nonnegative edge weights w. Now suppose each edge weight is increased by 1: $w'(u, v) = w(u, v) + 1$ for each $(u, v) \in E$. Let u, v be two vertices in G. **Claim:** If P is a shortest path connecting u and v in G with edge weights w, it must also be a shortest path connecting u and v in G with edge weights w'.

4. The setting is the same as 3) above. **Claim:** If T is a minimum spanning tree of G with edge weights w, then it must also be a minimum spanning tree of G with edge weights w'.

Problem 2: : Dynamic Programming [22 points]

a) [16 points] Here is the game called Math Effect Black Eye. You are Commander Shepherd and you are given a map of the galaxy: an $n \times n$ grid $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$. You are also given a positive integer $\mathsf{SV}[i, j]$ for each point (i, j) in the grid that represents the Strategic Value of (i, j). All the numbers $\mathsf{SV}[i, j]$ are distinct. You start at $(1, 1)$, the lower-left corner of the galaxy. For each round, you can either move right or up: Denote the current point by (i, j), then you can move to either $(i + 1, j)$ or $(i, j + 1)$. (Of course, you are not allowed to move outside of the galaxy.) For some unknown reason (classified), you can only move from one point to another (right or up for one step) if the latter has a higher SV. So a move from (i, j) to (i', j') is feasible only if $\mathsf{SV}[i', j'] > \mathsf{SV}[i, j]$. When you stop, the total SV of all points visited will be calculated, including $(1, 1)$ and the point where you stop. Use dynamic programming to compute the maximum total SV achievable by any feasible route from $(1, 1)$. Your algorithm only needs to output the maximum total SV. No need to output an optimal route for Commander Shepherd. No need to justify its correctness. What is the running time of your algorithm?

b)* [6 points]: In Math Effect Milky Way, Commander Shepherd can move to any of the four neighbors due to the advance of space jump technology (though he still cannot move outside of

the galaxy). For the same unknown reason (classified), he can only move from one point to another if the latter has a higher SV. Can you still use dynamic programming to help Commander Shepherd determine the maximum total SV achievable from $(1, 1)$? (Hint: Sort the points first.) Describe your algorithm and give its running time. No justification is needed.

Problem 3: Single-Source Shortest Paths [18 points]
Let $G = (V, E)$ be a connected, directed graph with nonnegative edge weights w, let s and t be two vertices of G, and let H denote a subgraph of G obtained by deleting some edges $E' \subseteq E$ from E. Suppose we want to reinsert exactly one edge from E' back into H, so that the shortest path from s to t in the resulting graph is as short as possible.

a) [8 points] Describe an algorithm based on the "brute-force" strategy: try edges in E' one by one and find the best one to reinsert (or maybe none of them really helps, in which case reinserting any edge in E' is fine). What is the running time? No justification is needed.

b)* [10 points] Describe an algorithm that runs in $O(|E| \log |V|)$ time. (Hint: Start by **making two calls** to single-source shortest paths.) No justification is needed.

Problem 4: Maximum Flow [18 points]
Consider a directed graph $G = (V, E)$ in which every $(u, v) \in E$ has a positive capacity $c(u, v)$. Let $s \in V$ be the source and $t \in V$ be the sink. Without loss of generality, below we always assume that t is reachable from s in G, so the value of a maximum flow from s to t is positive.

a) [6 points] An edge $(u, v) \in E$ is called *upward critical* if increasing the capacity $c(u, v)$ of (u, v) *strictly* increases the value of a maximum flow. Give an example in which G does not have any upward critical edge.

b)* [6 points] An edge $(u, v) \in E$ is called *downward critical* if decreasing the capacity $c(u, v)$ of (u, v) *strictly* decreases the value of a maximum flow. Prove that G always has at least one downward critical edge. (Remember that t is reachable from s in G.)

c) [6 points] Given G and a maximum flow f in G from s to t, describe an algorithm that finds a downward critical edge in time $O(|V| + |E|)$. No justification is needed.

Problem 5: NP-completeness [18 points]
Given a finite set $U = \{1, \ldots, n\}$ and a collection of subsets $S_1, \ldots, S_m \subseteq U$ (which are not necessarily disjoint), we call $C \subseteq [m]$ a *set-cover* if the union of S_i, $i \in C$, is U. For example, if $U = \{1, 2, 3, 4, 5\}$ and $S_1 = \{1, 2, 3\}$, $S_2 = \{2, 4\}$, $S_3 = \{3, 4\}$, $S_4 = \{3, 4, 5\}$, then $\{1, 4\}$ is a set-cover as $U = S_1 \cup S_4$; $\{1, 2, 3\}$ is not a set-cover since $S_1 \cup S_2 \cup S_3 = \{1, 2, 3, 4\}$ and 5 is missing. SET-COVER is the following decision problem: Given a finite set $U = \{1, \ldots, n\}$, a collection of subsets $S_1, \ldots, S_m \subseteq U$ and an integer k, decide if there is a set-cover C of size at most k (i.e., decide if one can pick at most k sets from $S_1, \ldots, S_m$ so that their union is U).

a) [6 points]: Show that SET-COVER is in NP.

b) [12 points]: Show that SET-COVER is NP-hard. (Hint: VERTEX-COVER.) Describe your reduction and prove its correctness. **Make sure the direction of reduction is correct!**