CS 344: Design and Analysis of Computer Algorithms **Rutgers: Fall 2019**

Practice Final Exam

December 10, 2019

Name: ______________________________ *NetID:* ______________

Instructions

1. Do not forget to write your name and NetID above.
2. The exam contains 5 problems worth 100 points in total *plus* one extra credit problem worth 10 points. You have 180 minutes to finish the exam. The exam is closed-book and closed notes.
3. **Note that problems appear on both odd and even numbered pages.** There should be more than enough space to write down your solution for each problem below the problem itself. But if you ran out of space, you can also use the extra sheet at the end of the exam; if you do so, be clear about which problem you are solving.
4. Remember that you can leave a problem (or parts of it) entirely blank and receive 25% of the grade for that problem (or part). However, this should not discourage you from attempting a problem if you think you know how to approach it as you will receive partial credit more than 25% if you are on the right track. But keep in mind that if you simply do not know the answer, writing a very wrong answer may lead to 0% credit.

   The only **exception** to this rule is the extra credit problem: you do not get any credit for leaving the extra credit problem blank, and it is harder to get partial credit on that problem.
5. **You should always prove the correctness of your algorithm and analyze its runtime.** Also, as a general rule, avoid using complicated pseudo-code and instead explain your algorithm in English.
6. You may use any algorithm presented in the class as a building block for your solutions.

Suggestion: Leave the extra credit problem for last as it worths fewer points.

Problem. #	Points	Score
1	20	
2	20	
3	20	
4	20	
5	20	
6	+10	
Total	100 + 10	

Problem 1.

(a) Mark each of the assertions below as True or False (you do *not* need a proof for this part).

(i) If $f(n) = n^{\log n}$ and $g(n) = 2^{\sqrt{n}}$, then $f(n) = \Omega(g(n))$. **(2 points)**

(ii) If $T(n) = 2T(n/2) + O(n^2)$, then $T(n) = O(n^2)$. **(2 points)**

(iii) If a problem in NP can be solved in polynomial time, then all problems in NP can be solved in polynomial time. **(2 points)**

(iv) If P = NP, then all NP-complete problems can be solved in polynomial time. **(2 points)**

(b) Prove or disprove the following assertions (you do need a proof or a counter example for this part).

(i) Consider a flow network G and a maximum flow f in G. There is always an edge e with $f(e) = c_e$, i.e., the flow passing the edge is equal to the capacity of the edge, such that increasing the capacity of edge e in G increases the maximum flow of the network. **(6 points)**

(ii) Suppose $G(V, E)$ is an undirected connected graph such that for every cut $(S, V - S)$, there are at least two cut edges in G. Then, every vertex v of G belongs to some cycle. **(6 points)**

Problem 2. You are given a two arrays $score[1:n]$ and $wait[1:n]$, each consisting of n positive integers. You are playing a game with the following rules:

- For any integer $1 \leq i \leq n$, you are allowed to pick number i and obtain $score[i]$ points;
- Whenever you pick a number i, you are no longer allowed to pick the previous $wait[i]$ numbers, i.e., any of the numbers $i-1, \ldots, i-wait[i]$.

Design an $O(n)$ time dynamic programming algorithm to compute the maximum number of points you can obtain in this game. It is sufficient to write your specification and the recursive formula for computing the solution (i.e., you do *not* need to write the final algorithm using either memoization or bottom-up dynamic programming).

(a) *Specification of recursive formula for the problem (in plain English)*: **(5 points)**

(b) *Recursive solution for the formula:* **(5 points)**

(c) *Proof of correctness of the recursive formula:* **(5 points)**

(d) *Runtime analysis:* **(5 points)**

Problem 3. You are given a directed graph $G(V, E)$ such that every *edge* is colored red, blue, or green. We say that a path P_1 is *lighter* than a path P_2 if one of the conditions below holds:

- P_1 has fewer number of red edges;
- P_1, P_2 have the same number of red edges and P_1 has fewer blue edges;
- P_1, P_2 have the same number of red and blue edges and P_1 has no more green edges than P_2.

Design an $O(n + m \log m)$ time algorithm that given G and vertices s, t, finds one the lightest paths from s to t in G, namely, a path which is lighter than any other s-t path. **(20 points)**

Problem 4. You are given an undirected *bipartite* graph G where V can be partitioned into $L \cup R$ and every edge in G is between a vertex in L and a vertex in R. For any integers $p, q \geq 1$, a (p, q)*-factor* in G is any subset of edges $M \subseteq E$ such that no vertex in L is shared in more than p edges of M and no vertex in R is shared in more than q edges of M.

Design an $O(m \cdot n \cdot (p + q))$ time algorithm for finding the *largest* (p, q)-factor of any given bipartite graph.

(20 points)

Hint: This problem is quite similar to the maximum matching problem we studied in the lectures.

Problem 5. Prove that the following problems are NP-hard.

(a) **Two-Third 3-SAT Problem:** Given a 3-CNF formula Φ (in which size of each clause is *at most* 3), is there an assignment to the variables that satisfies at least $2/3$ of the clauses? **(10 points)**

Hint: Use a reduction from the *3-SAT problem*. Recall that in the 3-SAT problem, we are given a 3-CNF formula Φ and the goal is to output whether there exist an assignment that satisfies *all* clauses.

(b) **Negative-Weight Shortest Path Problem:** Given a graph $G(V, E)$, two vertices s, t and *negative* weights on the edges, what is the weight of the shortest path from s to t? **(10 points)**

Hint: Use a reduction from the *s-t Hamiltonian Path problem.* Recall that in the s-t Hamiltonian Path problem, we are given a graph $G(V, E)$ and two vertices s, t and the goal is to decide whether there is a s-t path in G that passes through all other vertices.

Problem 6. [**Extra credit**] You are given a set C of p courses that a student has taken. You are also given q subsets $S_1, \ldots, S_q$ of C, and q integers $r_1, \ldots, r_q$. In order to graduate, the student has to meet the following q requirements:

- For every $1 \leq i \leq q$, the student needs to have taken at least r_i courses from the subset S_i.

The goal is to determine whether or not the courses taken by the student can be used to satisfy all q requirements. The tricky part is that any given course *cannot* be used towards satisfying multiple requirements.

For example, if one requirement states that the student should have taken 2 courses from $S_1 = \{A, B, C\}$ and another requirement asks for taking 2 courses from $S_2 = \{C, D, E\}$, then a student who had taken just $\{B, C, D\}$ *cannot* graduate.

Design a polynomial time algorithm that given a set C of p courses a student has taken, q subsets $S_1, \ldots, S_q$ of C, and q integers $r_1, \ldots, r_q$, determines whether or not the student can graduate.

Hint: Reduce this problem to a network flow problem on a carefully designed network.

Extra Workspace

Extra Workspace

Extra Workspace