Final Examination

CS 5114 (Fall 2018)

Assigned: December 4, 2018.
Due: submit PDF file containing your solutions on Canvas by 4pm on December 12, 2018.

Instructions

1. The Graduate Honor Code applies to this exam. In particular, you are not allowed to consult any sources other than your textbook, the slides on the course web page, your own class notes, and the instructor. Do not use a search engine.

2. You may consult the textbook, your notes, the course web site, or solutions to earlier homework assignments to solve the problems in the examination. Of course, I am available to answer your questions.

3. Describe your algorithms as clearly as possible. The style used in the book is fine, as long as your description is not ambiguous. Explain your algorithm in words. A step-wise description is fine. *However, if you submit detailed code or pseudo-code without an explanation, we will not grade your solutions.*

4. For every algorithm you describe, prove its correctness, and state and prove the running time of the algorithm. I am looking for clear descriptions of algorithms and for the most efficient algorithms and analysis that you can come up with. Except for one problem, I am not specifying the desired running time for each algorithm. I will give partial credit to non-optimal algorithms, as long as they are correct.

5. If you prove that a problem is $\mathcal{NP}$-Complete, remember to state the size of the certificate, how long it takes to check that the certificate is correct, and what the running time of the transformation is. All you need to show is that the transformation can be performed in polynomial time. Your transformation need not be the most efficient possible.

6. Do not make any assumptions not stated in the problem. If you do make any assumptions, state them clearly, and explain why the assumption does not decrease the generality of your solution.

7. You must also provide a clear proof that your solution is correct (or a counter-example, where applicable). Type out all the statements you need to complete your proof. *You must convince me that you can write out the complete proof. You will lose points if you work out some details of the proof in your head but do not type them out in your solution.*

8. You must prepare your solutions digitally, i.e., do not hand-write your solutions. I prefer that you use LaTeX to prepare your solutions. However, I will not penalise you if you use a different system. To use LaTeX, you may find it convenient to download the LaTeX source file for this document from the link on the course web site. At the end of each problem are three commented lines that look like this:

   ```
   % \solution{
   %
   % }
   ```

   You can uncomment these lines and type in your solution within the curly braces.

Good luck!

Problem 1 (24 points) Let us start with some quickies. For each statement below, say whether it is true or false or fill in the blanks. You do not need to provide a rationale for your solution.

1. If A and B are problems such that $A \in \mathcal{NP}$ and $B \leq_P A$, then $B \in \mathcal{NP}$.
2. In a directed graph, every edge has a positive integer weight that is bounded by the quantity w. The graph has n nodes and m edges. We can compute the shortest path between any two nodes in this graph in $O(w(n+m))$ time.
3. In a directed graph, let $d(x, y)$ denote the length of the shortest path between nodes x and y. If $e = (u, v)$ is an edge on a shortest path from s to t, then $d(s, t) = d(s, u) + d(u, t) - l(e)$, where $l(e)$ is the length of the edge e. Here, we are using the standard definition of the length of a path as the sum of the lengths of the edges in it.
4. If f is a maximum flow in a flow network and e is an edge such that the $f(e) = c(e)$, then e is an edge that crosses a minimum s-t cut.
5. If $\mathcal{P} = \mathcal{NP}$ and problem X is $\mathcal{NP}$-Hard, then there is a polynomial time algorithm to solve X.
6. In the proof for Bipartite Matching $\leq_P$ Network Flow, we created a flow network. The Ford-Fulkerson algorithm and the scaling algorithm run in the same time (asymptotically) on this flow network.
7. Every edge in a flow network has a capacity of one. The value of the maximum flow in this network is the _____ from s to t. (Fill in the blanks with a phrase.)
8. The place to be with good bread on campus is __ ___ ____.

Problem 2 (6 points) In class, we developed an algorithm that when given an undirected, unweighted graph G with n nodes and an integer k, checks if G has a vertex cover of size k or less in $O(k2^k n)$ time. Note that this time is polynomial in n (but exponential in k). Since we reduced Independent Set to Vertex Cover in class, we also have an algorithm that can check if a graph G has an independent set of size at least l (for some integer $l > 0$) in time polynomial in n. Is this statement true or false? Give a brief explanation for your answer.

Problem 3 (10 points) Consider the following problem: given an undirected, unweighted graph $G = (V, E)$ and a positive integer k, compute a spanning tree of G such that every node in the tree has degree $\leq k$, if such a tree exists. Extreme values of k are easy to handle. For example, if $k = 1$, then such a tree exists if and only G contains one edge (otherwise, the tree must have at least one vertex of degree two or more), Similarly, if $k \geq n - 1$, then any spanning tree of G will suffice. For a general (arbitrary) value of k, either develop a polynomial time algorithm for this problem or show that it is $\mathcal{NP}$-Complete.

Problem 4 (15 points) Given a directed graph $G = (V, E, l)$, where $l : E \to \mathbb{R}^+$ specifies a positive length for each edge in G, a source node s, and a sink node t, develop an algorithm to compute an integral flow[1] f such that the value $\nu(f) = 1$ and the quantity $\sum_{e \in E} l(e)f(e)$ is as small as possible. You can assume that there are no capacity constraints, i.e., each edge has unlimited capacity, but the flow along each edge cannot be negative. *Hint:* This is a "trick question" since it disguises another well-known problem in terms of flows.

Problem 5 (20 points) Consider the proof that Hamiltonian Cycle $\leq_P$ Travelling Salesman that we developed to prove that the second problem is $\mathcal{NP}$-Complete. We started with an undirected, unweighted graph $G = (V, E)$ that was an input to the Hamiltonian Cycle problem. We created an input for Travelling Salesman as follows: for every node $i \in V$, we created a city c_i, and for every pair of cities v_i and v_j, we defined $d(v_i, v_j) = 1$ if (i, j) is an edge in E and $d(v_i, v_j) = 2$ if (i, j) is not an edge in E. Consider the following modification: in the second case, i.e., (i, j) is not an edge in E, we set $d(v_i, v_j) = c$, where c is a constant ≥ 2. Let us denote this input to Travelling Salesman as $I(G, c)$. Answer the following questions.

[1] A flow f is *intgeral* if $f(e)$ is an integer for every edge e in E.

(a) (3 points) If G contains a Hamiltonian cycle, what is the length of the shortest travelling salesman tour in $I(G, c)$?

(b) (3 points) If G does not contain a Hamiltonian cycle, what is the length of the shortest travelling salesman tour in $I(G, c)$?

(c) (9 points) Suppose there is an α-approximation algorithm for the Travelling Salesman problem that runs in polynomial time, i.e., for every input to this problem, this algorithm computes a travelling salesman tour whose cost is at most α times the cost of the optimal tour. Determine a value of c so that you can use this approximation algorithm to distinguish unambiguously between the two cases: G has a Hamiltonian cycle and G does not have a Hamiltonian cycle. Explain your approach.

(d) (5 points) What can you conclude from these arguments?

Problem 6 (25 points) You are given a directed, node-weighted graph $G = (V, E, w)$ where $w : V \to \mathbb{R}$ is a weighting function that maps every node in V to a real number. We say that an edge (u, v) is *good* if $w(v) < w(u)$, i.e., if the weight of the head of the edge is less than the weight of the tail of the edge. Further, we say that a path is *good* if every edge in that path in good. Given two nodes s and t in V, develop an algorithm to compute the number of good s-t paths in G. Note that your algorithm should (implicitly) consider all good paths that start at s and end at t (not just the shortest paths) and it should return how many such paths there are. I am only interesting in computing this number exactly and correctly. *Hint:* The number of s-t paths may be exponential in the number of nodes in G. Hence, if you tried to develop an algorithm that *explicitly* computes every good s-t path, then it will have an exponential running time. It may be helpful if you draw some examples and work out the answers for these examples by hand.