CS 341, Fall 2025 T. Brown, É. Schost

ASSIGNMENT 5

DUE: Tue Dec 2, 11:59 PM. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.

Please read `http://www.student.cs.uwaterloo.ca/~cs341` for general instructions and policies.

Note: All logarithms default to base 2 (i.e., $\log x$ is defined as $\log_2 x$). **Note:** "Giving" an algorithm means doing the four parts (i)–(iv) as described on the course web page. **Proofs are required unless we explicitly state otherwise.** If you think we made a mistake and a proof is not required, feel free to ask. You may use the **unit cost model unless otherwise stated**.

1. [10 marks] (6 marks, 2 marks, 2 marks) In the CLIQUE3 problem, we are given a graph $G = (V, E)$ with maximum degree 3 and a positive integer k; we must determine if G has a clique of size at least k or not. (A clique is a (sub)set of the graph's vertices that have all-to-all edges. That is, the subgraph defined by these vertices is a *complete* graph.)

   (i) Prove that CLIQUE3 $\in$ **NP**.

   (ii) Here's a claimed proof that CLIQUE3 is **NP**-complete. Explain why the argument is incorrect.

   > We showed in part (a) that CLIQUE3 is in **NP**.
   > We (will) know from lectures that CLIQUE is **NP**-complete.
   > We complete the proof by showing that CLIQUE3 $\leq_{\mathbf{P}}$ CLIQUE: Let F be the (trivial) algorithm that takes in a graph G with vertices of degree at most 3 and a parameter k, and leaves both as-is. The algorithm F runs in polynomial time and gives a transformation from inputs of the CLIQUE3 problem to inputs of the CLIQUE problem, and the answer to these inputs is always identical. Therefore, this is a valid polynomial-time reduction and CLIQUE3 is **NP**-complete.

   (iii) In the VERTEXCOVER3 problem, we are given a graph $G = (V, E)$ with maximum degree 3 and a positive integer k; we must determine if G has a vertex cover of size at most k or not. It is known that VERTEXCOVER3 is **NP**-complete, and for this question we may use this fact without proof.

   Here's another claimed proof that CLIQUE3 is **NP**-complete. Explain why the argument is incorrect.

   > We already showed in part (a) that CLIQUE3 is in **NP**.
   > As stated in the question, we know that VERTEXCOVER3 is **NP**-complete.
   > We complete the proof by showing that VERTEXCOVER3 $\leq_{\mathbf{P}}$ CLIQUE3: Let F be the algorithm that transforms the input (G, k) into the input $(\overline{G}, n - k)$ where $\overline{G}$ is the complement of G—the graph on the same set V of vertices as G where for every pair of distinct vertices $u, v \in V$, (u, v) is an edge in $\overline{G}$ if and only if it is not an edge in G.
   > The algorithm F has polynomial-time complexity. And there is a vertex cover of size $\leq k$ in G if and only if there is a clique of size at least $(n-k)$ in $\overline{G}$. Therefore this is a valid polynomial-time reduction and CLIQUE3 is NP-complete.

2. [10 marks] Minimum S-spanning tree.

   Suppose we take as input an undirected graph $G = (V, E)$ with n nodes and m edges, and a non-negative weight function w such that $w(e)$ is the weight of edge e, as well as a set $S \subseteq V$ of **important nodes**. We want to connect all of the important nodes in $S \subseteq V$ together with the lowest total cost (i.e., lowest total sum of weights).

   At first glance this might sound like the minimum spanning tree problem, but note that we do not have to connect *all* nodes, but rather only the nodes in S. That said, it may be cheaper to connect nodes in S by using some nodes that are **not** in S.

   As a trivial example, if we have three nodes a, b, c and edges $(a, b, 1), (b, c, 1), (a, c, 10)$, and $S = \{a, c\}$, we could connect the important nodes a and c with a single edge at cost 10, or with two edges via a non-important node b at cost 2. The latter is, of course, better, even though we have also connected a non-important node.

   We can define two flavours of this problem: a **decision** version, and an **optimization** version.

   Decision version (MSST-Dec): In addition to the basic input above, the input also includes a threshold k. The desired output is: true if there is a way to connect all nodes in S with total cost $\leq k$, and false otherwise.

   Optimization version (MSST-Opt): The input is just the basic input above (**no** threshold k). The desired output is a **list of edges** that connect all nodes in S with the lowest total cost. If there is no way to connect them, you can return ∞.

   The goal in this question is to find a **polynomial Turing reduction** from the optimization version to the decision version. In other words, you are to prove MSST-Opt $\leq_P^T$ MSST-Dec.

   (To be clear, this involves "giving" an algorithm. So, you'll want to prove correctness, i.e., that your reduction always returns the *correct* value. And you'll want to compute your runtime, input size, and show the runtime is polynomial in the input size. Unit cost model is fine.)

3. [25 marks] *NP-Completeness* Consider the SINK-SOURCE SUBGRAPH (SSS) problem:

   Input: a directed graph $G(V, E)$.

   Output: "YES" iff there is a subgraph $H(V, E')$, with $E' \subseteq E$, such that each $v \in V$ satisfies one of two conditions: (i) either, in-degree(v) = 0 and out-degree(v) > 0; or (ii) out-degree(v) = 0 and in-degree(v) > 0. The in/out degree conditions are within H.[1]

   Let us call a subgraph H satisfying the above property *sink-source* subgraph. Observe that in a sink-source subgraph, there are no paths of length 2, since every vertex either has zero in-degree or zero out-degree.

   (a) [5 marks] Prove that SSS is in NP.

   (b) [20 marks] Prove that SSS is NP-Complete through a reduction from 3SAT. Remember to have an iff argument in your reduction.

   [Hint: Consider having one vertex u_i for each clause C_i in the 3SAT formula. Similarly have one vertex v_j for each variable x_j in the 3SAT formula. If x_j appears in C_i have

[1] In other words, an edge from a node $u \in H$ to a node $v \notin H$ does **not** count towards out-degree(u), and an edge from v to u does **not** count towards in-degree(u).

an edge from v_j to u_i. If $\overline{x_j}$ appears in C_i, add another edge from u_i to v_j. Add another vertex t to the graph and add edges from each v_j to t. Finally, add one more node s and add an edge from s to t.]