CMSC 441: Unit 5 Homework Solutions

December 9, 2017

(34.1-1) Suppose LONGEST-PATH-LENGTH(G, u, v) can be solved in polynomial time. Then LONGEST-PATH(G, u, v, k) can be solved by by calling LONGEST-PATH-LENGTH to compute the longest path length l and returning "true" if $k \leq l$ and "false" if $k > l$. Since the additional work beyond calling LONGEST-PATH-LENGTH is constant time, LONGEST-PATH $\in P$.

Conversely, suppose LONGEST-PATH$(G, u, v) \in P$. We can solve LONGEST-PATH-LENGTH as follows: compute LONGEST-PATH(G, u, v, k) for $k = 0, 1, 2 \ldots, n^2$ where n is the number of vertices of G. There are fewer than n^2 edges in G, so the longest path length must be less than n^2. Let k_0 be the first value of k for which the answer to LONGEST-PATH$(G, u, v, k_0 + 1)$ is "false"; then k_0 must be the longest path length. Note that this computation makes polynomially-many calls to a polynomial-time subroutine, and therefore runs in polynomial time.

(34.1-5) Suppose an algorithm makes c calls to a subroutine with running time $p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$ and performs $q(n) = b_l n^l + b_{l-1} n^{l-1} + \cdots + b_1 n + b_0$ additional work. Then the total running time is

$$c \cdot p(n) + q(n)$$

which is a polynomial of degree $\max(k, l)$. Now, suppose we have a sequence of subroutines s_i, $i \geq 1$, such that s_i has running time $\Theta(n^i)$. Furthermore, suppose our program, with input of size n, calls each of the first n subroutines $s_1, s_2, \ldots, s_n$. Certainly n is a polynomial, so our program makes polynomially many calls to subroutines with polynomial running times. However, the running time for the program is

$$\Theta\left(\sum_{j=1}^{n} n^j\right) = \Theta\left(\frac{n^{n+1} - 1}{n - 1}\right) = \Theta(n^n),$$

which is worse than exponential.

(34.2-1) First, recall the definition of *graph isomorphism*. For two graphs G and H with vertex sets $V(G)$ and $V(H)$, the map

$$f : V(G) \to V(H)$$

is an *isomorphism* if

1. f is one-to-one (injective).
2. f is onto (surjective).
3. (u, v) is an edge in G if and only if $(f(u), f(v))$ is an edge in H.

If $f : V(G) \to V(H)$ is an isomorphism, we say that G and H are *isomorphic*.

Let us assume the language encodes $x = \langle G_1, G_2 \rangle$ in adjacency list form: for each of n vertices u, we list the vertices v such that (u, v) is an edge, so each graph is represented by n lists, some of which may be empty (note that both G_1 and G_2 must have the same number of vertices n if they are to be isomorphic). Each vertex can be represented by the binary encoding of a number between 1 and n. Then a certificate consists of a mapping $f : V(G_1) \to V(G_2)$, which could be encoded as a permutation of the vertices of G_2; that is, if the vertices of G_2 are $(v_1, v_2, \ldots, v_n)$, then the map is encoded as $(v_{f(1)}, v_{f(2)}, \ldots, v_{f(n)})$. To make the description of the verifier algorithm simpler, we will also include an encoding of the inverse map $f^{-1} : V(G_2) \to V(G_1)$ in the certificate; the inverse must exist if f is one-to-one and onto, and including it only doubles the size of the certificate. Note that since the encoding x of G_1 and G_2 consists of $2n$ lists of vertices, the size of the certificate is $O(|x|)$.

Now, to verify the certificate, we need to do the following:

1. Check that $|V(G_1)| = |V(G_2)| = n$. This requires scanning the encoding of G_1 and G_2, but since each has at most $O(n^2)$ edges, this can be done in $O(n^2)$ time.
2. Check that the table representations of f and f^{-1} are both n-long. This is simply a linear scan of the tables to check their lengths.
3. Check that f and f^{-1} are both permutations of the integers 1 to n, which can be done simply by checking that each value occurs exactly once in each table. This can also be accomplished with a linear scan of each table, and could be combined with step one.
4. Verify that f and f^{-1} are in fact inverses, which can be done by looping over the vertices v, labelled 1 to n, and checking that $f(f^{-1}(v)) = f^{-1}(f(v)) = v$. This requires a single linear scan through the values 1 to n.
5. For all edges $(u, v) \in E(G_1)$, verify that $(f(u), f(v)) \in E(G_2)$ and for all edges $(s, t) \in E(G_2)$, verify that $(f^{-1}(s), f^{-1}(t)) \in E(G_1)$. This requires looping over all edges of G_1 and G_2, which is, at worst, $O(n^2)$.

This entire algorithm runs in time $O(n^2)$, which is certainly polynomial in $|x|$ since the encoding of an n vertex graph in adjacency list form is more than n bits long.

(34.2-6) To show that the decision problem HAM-PATH is in NP, we need to show that there are certificates whose size is polynomial in the encoded input size n and that there is a verifier function that can verify a certificate in time polynomial in n. A certificate for HAM-PATH is a list of vertices, in order, that describe the hamiltonian path from u to v in G:

$$u = u_1, u_2, \ldots, u_k = v.$$

Since the consecutive pairs of vertices in the list represenent distinct edges in G, the encoding of the certificate is *smaller* than the encoding of the input $\langle G, u, v \rangle$, which must encode all the edges of G. To verify a certificate, we must loop over the list of vertices and check the following:

1. $u_1 = u$
2. (u_i, u_{i+1}) is and edge in G for $i = 1, 2, \ldots, k-1$
3. $u_k = v$
4. Each vertex of G is listed exactly once

Since these conditions define a hamiltonian path, there are no "false" certificates that could convince us the answer to an instance of HAM-PATH is "Yes" when it is in fact "No." The work to verify a certificate is dominated by step (2), checking that each consecutive pair of vertices is an edge in G. However, for any single pair (u_i, u_{i+1}), checking that it is an edge is simply checking an adjacency list or adjacency matrix, which can be done in $O(|V|)$ time ($O(1)$ for adjacency matrix, but adjacency list could require scanning up to $|V|$ outgoing edges from u_i), and this must be done $|V|$ times, once per vertex in the path. To determine that each vertex is visited exactly once, we just tally which vertices are visited, and how many times, as we loop over the list; then it is a matter of looping over the list of counts and checking that each entry is one. All together, the time to verify a certificate is $O(|V|^2)$, and, if the answer is "Yes", there are at least $|V| - 1$ edges which must be encoded in the input, so the running time is polynomial in n.

(34.3-1) Suppose the circuit is satisfiable; then the final AND gate must output 1, so all of its inputs must be 1. Tracing back from the AND through the bottom-most wire, we encounter another AND gate, for which all the inputs must be 1. We can conclude that $x_3 = 0$ since it will be inverted to 1 by the bottom-left NOT gate. Now, starting from $x_3 = 0$, we see that the input to the second NOT gate is 1, so it's output is 0; it's output is one of the two inputs to the two-input AND gate, and so the output of the two-input AND is also 0. Since this 0 feeds into the final three-input AND, we must have that the output of the circuit is 0. But this is a contradiction. Therefore the circuit is not satisfiable.