Fundamental Algorithms, Assignment 12

SOLUTIONS

1. Which of the following problem classes are in P and which are probably not in P. (By probably not we mean that we do not as of today know that it is in P but of course tomorrow somebody might come up with a clever algorithm.)

   (a) `PRIME`. The input here would be integers n and Yes would be returned iff n is prime.
   `Solution:`In P via the Agarwal, Kayal, Sexana algorithm.

   (b) I gave the above problem twenty years ago. What was the answer then?
   `Solution:`Probably not in P. (The above algorithm was discovered in 2002.)

   (c) `CONNECTED-GRAPH`. The input here would be a graph G and Yes would be returned iff the graph was connected.
   `Solution:`In P as we can use, for example, Breadth-First Search.

   (d) `TRAVELING-SALESMAN`. The input here would be a graph G together with a positive integer weight $w(e)$ for each edge e and an integer B. Yes would be returned iff there was a Hamiltonian Cycle which had total weight at most B.
   `Solution:`Probably not in P. This is a big open question.

   (e) `SPANNING-TREE`. The input here would be a graph G together with a positive integer weight $w(e)$ for each edge e and an integer B. Yes would be returned iff there was a spanning tree which had total weight at most B.
   `Solution:`In P as we can use Kruskal's or Prim's algorithm.

   (f) `ALMOSTDAG`. The input here would be a directed graph G. Yes would be returned iff there was a set of at most 10 edges of G that could be removed from G so that the remaining graph is a `DAG`. (Your argument should work with 10 replaced by any *constant* value.)
   `Solution:`In P. For *every* set of 10 edges – and there are $O(n^{20})$ of them, apply TopSort to see if G is a `DAG` after their removal. Each instance of TopSort is polynomial (certainly $O(n^3)$ with "room to spare") so multiplying by the number of instances gives $O(n^{23})$ which is still polynomial. (Note that this argument does not work if 10 is replaced by, say, $\lfloor\sqrt{n}\rfloor$ as then you would have $n^{\sqrt{n}}$ which is not polynomial.)

2. Show that the following problem classes are in NP. (That is, describe the certificate that the Oracle gives and describe the procedure that Verifier will take. Warning: Do not trust Oracle! For example, if Oracle gives you n distinct vertices you have to verify that they are indeed distinct!)

   (a) `PRIME-INTERVAL` The input here would be integers n, a, b. Yes would be returned iff there was a prime p which divided n and for which $a \leq p \leq b$.
   `Solution:`Oracle gives p. Verifier checks that $a \leq p \leq b$, that $p|n$ and that p is prime, the last using the Agarwal, Kayal, Sexana algorithm.

   (b) `TRAVELLING-SALESMAN` As described above.
   `Solution:`Oracle gives the ordering $x_1, \ldots, x_n$ of the vertices. Verifier must check that these are distinct vertices, that they are all the vertices, and that the sum of the weights of the edges $\{x_i, x_{i+1}$ (including $\{x_n, x_1\}$ is at most B.

   (c) `COMPOSITE` The input here would be an integer n. Yes would be returned if n was composite. For this problem I want two solutions. One (the easier one) uses the Agarwal, Kayal, Saxena algorithm. The second should not use the Agarwal, Kayal, Saxena algorithm.
   `Solution:`One: Use AKS for Prime and then flip the Yes/No answer. That is, the Oracle is not needed at all. This is OK. Indeed any L which is in P is in NP since you don't need to use the Oracle. The other: Oracle gives a, b with $n = ab$. Verifier checks the multiplication.

   (d) 3-COLOR The input here would be a graph G. Yes would be returned if there was a three coloring of the vertices such that no two adjacent vertices v, w had the same color.
   `Solution:`Oracle gives the three coloring. Verifier checks that for every $w \in Adj[v]$, v, w do not have the same color.

   (e) `NEAR-DAG`. The input here would be a directed graph G and an integer B. Yes would be returned if there was a set of at most B edges that could be removed from G so that the remaining graph was acyclic. (This is like `ALMOST-DAG` with the critical distinction that B is not restricted to 10, or any constant value. Rather, B can depend on the number of vertices of G.)
   `Solution:`Oracle gives the B edges to be removed. Verifier counts

them, makes sure they are edges in the graph, and then removes them from G and applies TopSort to see if the remaining graph is indeed a DAG. Alternately to TopSort, Oracle could give the ordering $x_1, \ldots, x_n$ of the vertices such that all edges are "to the right". Then Verifier would have to check that these are indeed the n vertices with no repetition and that every edge does indeed go to the right.

3. For the following pairs L_1, L_2 of problem classes show that $L_1 \leq_P L_2$. That is, given a "black box" that will solve any instance of L_2 in unit time, create a polynomial time algorithm that will solve any instance of L_1 in polynomial time.

   (a) Let L_2 be `TRAVELLING-SALESMAN DESIGNATED PATH`. The input here would be a graph G, two designated vertices, a source v_1 and a sink v_n, together with a positive integer weight $w(e)$ for each edge e and an integer B. Yes would be returned iff there was a Hamiltonian Path (i.e., one goes through all the vertices $v_1, \ldots, v_n$ in some order (starting and ending at the designated vertices) but does *not* return from v_n back to v_1) which had total weight at most B. L_1 is `TRAVELLING-SALESMAN` as described above.
   `Solution:`For each edge $e = \{x, y\}$ of the graph ask L_2 if there is a Hamiltonian Path from x to y (that is, source x, sink y) whose length is at most $B - w(e)$. If you ever get a Yes then the answer to L_1 is Yes as you add the edge e to the Hamiltonian path. But if you always get No then the answer to L_1 is No as a Hamiltonian cycle of length $\leq B$ would have to use *some* $e = \{x, y\}$ and cutting it out would give a Hamiltonian path of length less that $B - w(e)$ with that source and sink.

   (b) Let L_2 be `CLIQUE`. The input here would be a graph G together with a positive integer B. Yes would be returned iff there was a clique with at least B vertices. (A set of vertices in a graph G is a clique if *every* pair of them are adjacent.) Let L_1 be `INDEPENDENT-SET`. The input here would be a graph G together with a positive integer B. Yes would be returned iff there was a independent set with at least B vertices. (A set of vertices in a graph G is an independent set if *no* pair of them are adjacent.)
   `Solution:`G has an independent set of size at least B if and only if G^c has a clique of size at least B. Here G^c is the *complement* of G, pairs of vertices being adjacent in G^c iff they are not adjacent

in G. Given G it takes time $O(n^2)$ to create G^c. Our algorithm for L_1 on G would be to create G^c and then apply L_2 to it, and that will return the correct answer to the original problem.

4. Assume `PRIME-INTERVAL` (defined above) is in P. Using it as a black box give a polynomial time algorithm with input integer $n \geq 2$ that returns some prime factor p. (Caution: This means polynomial in the number of digits in n, or what is sometimes called polylog n, meaning $O(\lg^c n)$ for some constant c.)
   `Solution:`We search for the prime factor by continually splitting the interval in half. We start that we know `PRIME-INTERVAL(n,2,n)` is true. Now check PRIME-INTERVAL(n,2,$\frac{n}{2}$). If true we know there is a prime in $[2, \frac{n}{2}]$, else we know there is a prime in $(\frac{n}{2}, n]$. We keep splitting the interval in half until we have an interval of length one where there is a prime. This takes $\lg n$ calls. (*) Further, give a polynomial time algorithm with input integer $n \geq 2$ that returns the entire prime factorization of n.
   `Solution:`Apply the above to get the first prime factor p and now iterate the entire procedure on $\frac{n}{p}$. Each time we get a prime the new value of n is at most half of the previous value so we do this at most $\lg n$ times, each time takes at most $\lg n$ calls, so tis would require at most $\lg^2 n$ calls, as desired.