Fundamental Algorithms - Spring 2018
Homework 12

Daniel Rivera Ruiz
Department of Computer Science
New York University
drr342@nyu.edu

1. (a) `PRIME`. In 2006, the AKS primality test algorithm was developed to determine whether a given number n is prime. When first published, the asymptotic complexity was given by $O(\log^{12}(n))$, which is polynomial in the number of digits of n. Since then the algorithm has been improved reducing this value to $O(\log^{6}(n))$. This means that `PRIME` is in P.
   (b) Twenty years ago the AKS primality test algorithm didn't exist yet, thus `PRIME` was not in P.
   (c) `CONNECTED-GRAPH`. This problem is in P, since it suffices to run `BFS` on G, which takes time $O(V + E)$: if `BFS` visits all the vertices in G return `True`, otherwise return `False`.
   (d) `TRAVELING-SALESMAN`. As of today, this problem is not in P.
   (e) `SPANNING-TREE`. This problem is in P: we need only to run Kruskal's algorithm on G to find the `MST`, which takes $O(E \log_2 V)$. If the weight of the tree is less or equal to B return `True`, otherwise return `False`.
2. (a) `PRIME-INTERVAL`.
   - Certificate: the prime number p.
   - Verification: divide n by p in $O(\log^2(n))$ time, check that $a \leq p \leq b$ in $O(1)$ and check that p is prime using `AKS`.

   (b) `TRAVELING-SALESMAN`.
   - Certificate: the ordered set of vertices $V = \{v_1, v_2, \ldots, v_n\}$.
   - Verification: check that V has no duplicates in $O(n)$ to confirm that the set forms a Hamiltonian path. Furthermore, check that there is a back edge in $O(1)$ to confirm that the vertices form a Hamiltonian cycle. Finally, add the weights of all the edges in $O(n)$ to confirm that the sum is less or equal to B.

   (c) `RAMANUJAN`.
   - Certificate: the two couples of numbers $C_1 = \{a, b\}$ and $C_2 = \{c, d\}$.
   - Verification: find the cube of each number with two multiplications in $O(m^4)$, where m is the number of digits in the number. Perform the two additions in $O(m)$. Check that $n = a^3 + b^3$ and $n = c^3 + d^3$ in $O(1)$.

   (d) `COMPOSITE`.
   - Certificate: the prime factorization of $n = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_m^{\alpha_m}$.
   - Verification 1: ignore the oracle's certificate and run `AKS` to test for primality.
   - Verification 2: perform the exponentiations and multiplications in polynomial time and check for equality with n in constant time.

   (e) `3-COLOR`.
   - Certificate: a set of tuples (v, c) for all the vertices v in G with the assigned color c.
   - Verification: Traverse the adjacency lists for all v in $O(E)$ and check that for each vertex no adjacent vertex has the same color as itself.
3. (a) Let's consider a vertex w and its adjacency list $adj(w) = \{w_1, w_2, \ldots, w_m\}$. We run `TRAVELLING-SALESMAN DESIGNATED PATH` with $v_1 = w$ and $v_n = w_i$ for $1 \leq i \leq m$: if any of the runs returns `True`, we calculate the total weight of all the edges in the path plus the weight of the edge $\{w, w_i\}$ (which must exist since the vertices selected are adjacent): if the

total weight is less or equal to B we return `True`. We repeat this procedure for all vertices in G until we find a Hamiltonian cycle. If we traverse all vertices without success, we return `False`.
Assuming that L_2 - `TRAVELLING-SALESMAN DESIGNATED PATH` can be solved in unit time, the previous algorithm can solve L_1 - `TRAVELING-SALESMAN` in polynomial time, since we will run L_2 at most $O(E)$ times, which is the upper bound for the sum of the lengths of all the adjacency lists in the graph. All the other calculations in the algorithm (additions, comparisons, etc.) can also be done in polynomial time.

4. If we consider `PRIME-INTERVAL` with $a = 2$ and $b = n$ we can guarantee that it will return `True`, since all the prime factors of n must be in this interval. With this initialization, we will consider intervals of half the size at each iteration of the algorithm: in the first step we will evaluate `PRIME-INTERVAL`$(a = 2, b = \frac{n}{2}, n = n)$. If this returns `True`, we know there is a prime in the interval $[2, \frac{n}{2}]$, otherwise there must be a prime in $[\frac{n}{2}, n]$.
We will repeat this procedure always selecting the interval where the prime factor p lies until we get two intervals of size one. At this point we have found a prime factor of n, since `PRIME-INTERVAL` must return `True` for (at least) one of the intervals.
Given the assumption that `PRIME-INTERVAL` is in P and the fact that the algorithm makes $\log_2(n)$ calls to this procedure, we can conclude that the overall time complexity of the algorithm is also in P, particularly of the form $O(log_2(n) \cdot PI(n))$, where $PI(n)$ is the (polynomial) time complexity of `PRIME-INTERVAL`.

5. After `DFS-VISIT[`v`]` is completed, we will have the following conditions:
   - All the vertices that are reachable from v and v itself will be black. At the beginning of the algorithm, v is painted gray and then we call `DFS-VISIT` recursively on all the vertices that are white and adjacent to v. At the end of all the recursive calls, we start painting all the found vertices black. Since v is at the top of the recursion, all the vertices that where gray at some point during the algorithm must be black at the end.
   - All the vertices that are not reachable from v will be white. These vertices were never touched during the recursive calls to `DFS-VISIT`, and therefore remain white (as per the problem definition at the beginning we *"assume all vertices of G have color white"*).
   - There will be no gray vertices. The only condition under which there can be gray vertices at the end of a call to `DFS-VISIT`, is when this call was made recursively from within another `DFS-VISIT`. In that case, all the vertices that had been painted gray in the outer `DFS-VISIT` will still be gray. However, we know that this is not the case when we call `DFS-VISIT[`v`]` because as per the problem definition at the beginning we *"assume all vertices of G have color white"*.