Basic Algorithms, Assignment 13

Solutions

1. 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.

2. Suppose that we are doing Dijkstra's Algorithm on vertex set $V = \{1, \ldots, 500\}$ with source vertex $s = 1$ and at some time we have $S =$

$\{1,\ldots,100\}$. What is the interpretation of $\pi[v], d[v]$ for $v \in S$?
`Solution:` $d[v]$ is the minimal cost of a path from s to v and $\pi[v]$ will be the vertex just before v on that path.
What is the interpretation of $\pi[v], d[v]$ for $v \notin S$?
`Solution:` $d[v]$ is the minimal cost of a path $s, v_1, \ldots, v_j, v$ where all the $v_1, \ldots, v_j \in S$. $\pi[v]$ will be the vertex just before v in this path, here v_j.
Which v will have $\pi[v] = NIL$ at this time.
`Solution:` Those v for which there is no directed edge from any vertex in S to v.
For those v what will be $d[v]$?
`Solution:` Infinity

3. (Extra from last week!) You may use Agarwal/Kayal/Saxena but, if so, mark clearly how it is used.

   (a) Call a positive integer n `XINYU` if it has at least one prime divisor p of the form $p = 10k + 7$. Show $XINYU \in NP$.
   `Solution:` Oracle gives the value p. Verifier must check that

      i. p divided by 10 gives a remainder of 7
      ii. p is prime – using AKS
      iii. p divides n

   (b) (harder!) Call a positive integer n `YUCHEN` if it has exactly one prime divisor p of the form $p = 10k+7$. Show $YUCHEN \in NP$.
   `Solution:` Oracle gives the prime factorization (possibly with repetition) $n = p_1 \cdots p_r$ with p_1 of the form $10k + 7$. Verifier must check that

      i. p_1 divided by 10 gives a remainder of 7
      ii. All other p_i divided by 10 do not give a remainder of 7
      iii. All p_i are prime – using AKS.
      iv. $n = p_1 \cdots p_r$

   (Note: As all $p_i \geq 2$ the number of factors $r \leq d$, d the number of digits of n. So if AKS takes $O(d^c)$ applying AKS to each factor takes $O(d^{c+1})$, still polynomial.)

4. Let G be a `DAG` on vertices $1, \ldots, n$ and suppose we are *given* that the ordering $1 \cdots n$ is a Topological Sort. Let `COUNT[i,j]` denote the number of paths from i to j. Let s, a "source vertex" be given. Give an efficient algorithm to find `COUNT[s,j]` for all j.

`Solution:` Lets assume $s = 1$ (we can ignore the earlier vertices, if any) and write $COUNT[j]$ for $COUNT[1, j]$. We set $COUNT[1] = 1$. The key is that $COUNT[1, j]$ is the sum, over all $i < j$ with i, j a directed edge, of $COUNT[1, i]$. Why? Well, every path from 1 to j will have a unique penultimate point $i < j$ and given i there will be precisely $COUNT[i]$ such paths. One way to implement this is to make a reverse adjacency list, create for every j a list $Adjrev[j]$ of those i with a directed edge from i to j. This can be done in time $O(E)$ by going through the original adjacency lists and when $j \in Adj[i]$ adding i to $Adjrev[j]$. Then we can implement this sum. The total time (assuming addition takes unit time) is $O(E)$.