CSE 551 Homework 6 Solutions

December 2021

1 Question 1

Consider the problem of determining, given an undirected graph G and an integer k as input, if there is a path in G with at least k vertices (each vertex occurs at most once in the path). Either give a polynomial time algorithm to solve the problem, or show it is NP-complete.

Solution: Call the path LONGPATH. The problem is in NP. Consider the following algorithm:

$M =$ "On input $\langle G, P\rangle$: // G is the graph, P is the certificate.
1. Step through P and verify it is a path. If not reject.
2. If the length of P is $\geq k$ then accept. Else reject."

It is worth observing, that verifying our path can be a bit tricky. We will require the encoding of a path to be a sequence of vertices in a given order $v_1 v_2 \ldots v_k$ and in order to check if the path is valid, one must make a few (but still polynomial with respect to the input!) passes through the data to ensure that:

1. There are no duplicate vertices.
2. All v_i and v_{i+1} in fact have an edge in between them in the parent graph.

Notice that HAMPATH reduces to this problem. HAMPATH asks for a path that uses each vertex. So, given any instance of HAMPATH, count the number of vertices in the graph and ask LONGPATH if there is a path with at least $|G|$ vertices.

2 Question 2

Consider the problem of determining, given a directed, acyclic graph G and an input integer k as input, if there is a path in G with at least k vertices (each vertex occurs at most once in the path). Either give a polynomial time algorithm to solve the problem, or show it is NP-complete.

Solution: This is actually a bit different from Problem 1 and the problem is, in fact, in P. To solve the problem:

$M =$ "On input $\langle \text{DAG } G, k\rangle$:
1. S $\leftarrow$ Topologically sort G.
2. For $v \in S$:
  a. If $\exists (u, v)$:
    i. $\text{len}(v) = \max_{(u,v)}\{\text{len}(u) + 1\}$.
  b. Else:
    ii. $\text{len}(v) = 0$.
3. Return $\max_{v\in V} \text{len}(v) \geq k - 1$?"

To see that the construction returns the correct result, recall that (1) any DAG may be topologically sorted, (2) in a topological ordering S, given an (u, v), u is listed before v. Thus, when (u, v), u is the parent of v. The idea behind the algorithm is to choose the parent node with the longest path (step i) in order to maximize the distance so far seen for v. Notice that $\text{len}(u)$ is always computed before v since u shows up before v in the sorting.

To see the algorithm runs in polynomial time, recall topological sorting takes $O(|V(G)|+|E(G)|)$ time. Thus, step 1 is still polynomial with respect to the input. Steps a and b require no more than polynomial time. Therefore as step 2 iterates over no more than a polynomial number of inputs, the entire algorithm is, in fact, polynomial.

3 Question 3

We say two graphs G and H are isomorphic when there is a one-to-one mapping from the vertices of G to the vertices of H so that edges map to edges and non-edges map to non-edges. Consider two graphs G_1 and G_2. Either give a polynomial algorithm to find whether G_1 contains a subgraph G' that is isomorphic to G_2 or show it is NP-complete.

Solution: Call this problem SUB-ISO. First we establish SUB-ISO $\in$ NP. Define the non-deterministic algorithm M to operate as follows:

M = "On input $\langle G = (V_1, E_1), H = (V_2, E_2)\rangle$:

1. Non-deterministically guess a subgraph $G' = (V_1', E_2')$.
2. Non-deterministically guess an isomorphism $f : V_1' \to V_2$.
3. For every $(u, v) \in G'$:
4. Ensure $(f(u), f(v)) \in H$, reject otherwise.
5. For every $(u, v) \in H$:
6. Ensure $(f^{-1}(u), f^{-1}(v)) \in G$, reject otherwise.
7. Accept."

A couple notes: (1) The isomorphism f may be guessed as it is a look up table. (2) Every isomorphism has an inverse. (3) Looking up values of f and f^{-1} does not take more than polynomial time. From this we conclude there is a non-deterministic polynomial time algorithm and that SUB-ISO is in NP.

To show the problem is NP-complete, you can reduce HAMPATH to this problem in polynomial time by taking any graph G, counting the number of vertices $|V|$, then create a graph H that has vertices $v_1, v_2, \dots v_{|V|}$. With edges $(v_1, v_2), (v_2, v_3), \dots (v_{|V|-1}, v_{|V|})$. If there is a Hamiltonian path it is a graph this isomorphic to H and therefore any instance of HAMPATH can be reduced to an instance of SUB-ISO.

4 Question 4

The Google autonomous car validation team uses a tool that gives the lines of code that are covered by a test case. They want to select the fewest test cases to validate the entire code. Either give a polynomial algorithm to find a set which has fewest test cases that cover the entire code or show it is NP-hard.

Solution: Call this problem TEST. Since the question asks to show if TEST is NP-hard (as opposed to NP-complete) we do not need to establish that this problem is in NP. We proceed to reduce SET-COVER to this problem.

Consider an instance of SET-COVER $\langle \mathsf{U}, \mathcal{S}, \mathsf{k}\rangle$, where U are the elements and $\mathcal{S}$ is the set of subsets of U. We can "transform" to an instance of TEST by treating each subset $S_i \in \mathcal{S}$ as a test that covers each "line of code" $s \in S_i$. Thus, we may run it through a solver for TEST and determine the fewest test cases needed to validate the code. After this, count the number of such sets and if this number is $\leq k$ accept, else reject.

5 Question 5

Consider an interval scheduling problem; each request has a set of intervals and there is a single processor to serve the requests, and an integer k is given. Either give a polynomial-time algorithm to determine whether the processor can serve at-least k requests or show it is NP-complete.

Solution: Run the Greedy Interval Scheduler to compute the maximal number of compatible intervals. Check if this $\geq k$, if so accept, else reject. Recall, the bottleneck the Greedy Interval Scheduler was from sorting which leaves this problem firmly in P.

6 Question 6

We have a puzzle that consists of a $p \times q$ grid of squares, where each square can be empty, filled with a red stone, or filled with a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) Every row contains at least one stone, and (2) No column contains stones of both colors. For some initial configurations of stones, reaching this goal might be impossible. Either give a polynomial-time algorithm to determine whether the puzzle can be solved, given an initial configuration of red and blue stones as input, or show that it is NP-hard.

Solution: Reduce 3-SAT to this problem. An instance of 3-SAT has p clauses $C_1 \dots C_p$ and q variables $x_1 \dots x_q$. Let every clause correspond to a row in the puzzle and every column correspond to a variable. More precisely, place blue/red stone in position (i, j) (read row *then* column like a matrix) iff C_i is evaluates to true when x_j is set to true/false. Call this board B.

Theorem 6.1. *(1) A solution to B gives a solution to the instance of* 3-SAT*. (2) No solution to B indicates there is no solution to the instance of* 3-SAT*.*

Proof. (1) Given a solution for B, every column has at most one color associated with it. Recall that each column "corresponds" to the truth assignment of a variable. If the column i has the color blue/red in it, let variable x_i be true/false. If there is no color, then arbitrarily assign the value of x_i (this variable was "redundant"). The corresponding set of truth assignments satisfy the original 3-SAT instance since every row has one pebble, this means that all clauses were satisfied and no variable x_i could be given two distince truth values as this would mean the column had both a red and blue stone in the end. From this interpretation (2) follows immediately as any construction that fills all rows requires at least one variable to take two truth assignments. □

Hence the problem NP-hard.