COMP 3721 Question Bank 4 Suggested Solutions

NP-completeness

1. (i) True.
   (ii) It's still unknown.
   (iii) True.
   (iv) It's still unknown.
   (v) True.

2. (i) False. Since $P \subseteq NP$, certainly some problems in NP can be solved in polynomial time.
   (ii) True. In order to be in NP, there must be a certificate which allows us to test membership in polynomial time. Thus a certificate is of polynomial length, and so we can enumerate all the certificates in exponential time and test each one using the verification procedure.
   (iii) Unknown. This is because it is still open whether $P = NP$.
   (iv) Unknown. If $P = NP$, this is correct. Otherwise it could be false.

3. Suppose that G is a graph that has a feedback vertex set of size k. The certificate is the set V' of vertices that are in the feedback vertex set. To determine whether V' is indeed a feedback vertex set, we need to test whether every cycle in G passes through at least one of these vertices. Note that there are exponentially many cycles in a graph, so testing each cycle would not run in polynomial time. Instead we delete all the vertices of V' from G, along with any incident edges. Let G' denote the resulting graph. Then we test whether G' has a cycle, by running DFS and checking whether the resulting tree has at least one backedge. If so, V' is not a feedback edge set (since the cycle in G' does not pass through any vertex of V') and otherwise it is.

4.

Subgraph-Isomorphism (SI) $\in$ NP

First we show that subgraph-isomorphism is in NP. The certificate consists of the mapping of the vertices of G_1 to the vertices of G_2, such that if there is an edge between any two vertices in G_1 then there is also an edge between the corresponding vertices in G_2. This mapping can be specified using a table T such that vertex i of G_1 maps to vertex $T(i)$ of G_2. Given such a table T, we can verify by checking for each edge $\{i, j\}$ of G_1, whether $f(i)$ and $f(j)$ are connected by an edge in G_2. If this is true, then A outputs yes. Otherwise, A outputs no. The time needed is at most $O(m_1 \cdot m_2)$, where m_1 and m_2 are the number of edges in G_1 and G_2 respectively. This is polynomial in the size of G_1 and G_2. Thus, the subgraph isomorphism problem is in NP.

Clique $\leq_P$ SI

Goal:

Next we want to show that Clique is polynomially reducible to the subgraph-isomorphism problem (SI). That is, we want a polynomial time computable function, which given an instance of Clique (a graph G and a positive integer k) outputs an instance of SI (two graphs G_1 and G_2) such that G has a clique of size k if and only if G_1 is a subgraph of G_2.

Transformation:

We are given an instance of Clique: graph G and integer k. Assuming k does not exceed the number of vertices in graph G, we construct G_1 to be the complete graph with k vertices in $O(k^2) = O(n^2)$ time, where n is the number of vertices in G, and construct G_2 to be the same as the graph G. Then we output G_1 and G_2 as the instance of the subgraph isomorphism problem. If, however, k exceeds the number of vertices in graph G, we just output any no-instance of the subgraph-isomorphism problem (because graph G obviously cannot have a clique of size exceeding the number of vertices in G). Since the reduction runs in polynomial time, the clique problem is polynomial-time reducible to the subgraph isomorphism problem.

Correctness:

Assume that k does not exceed the number of vertices in G. The correctness of the transformation follows, since G has a clique of size k if and only if the complete graph consisting of k vertices (i.e. G_1) is a subgraph of G (i.e. G_2).

If k is more than the number of vertices in G, then it is clearly a no-instance of the clique problem. In this case the transformation generates a no-instance of the SI problem and so it is correct.

5.

Set Cover (SC) $\in$ NP

Th certificate consists of the subset C of F. Given C, we first check that C contains k sets. This can be done in $O(k)$ time. Note that the input size is at least k since F contains at least k sets. Next, for each element $x \in X$, we check if x appears in some set in C. This takes $O(k|X|^2)$ time, since each set in C has at most $|X|$ elements. Since both checks can be done in time polynomial in the input size, we conclude that the set cover problem is in NP.

Vertex Cover $\leq_P$ Set Cover

Goal:

We want a polynomial time computable function, which given an instance of the VC problem (a graph G and integer k) produces an instance of the SC problem (a domain X, a family of sets F over X, and integer k') such that G has a vertex cover of size k if and only if F has a subfamily of k sets that covers X.

Transformation:

Given a vertex cover instance $G(V, E)$ and k. Construct the set X to be the set of edges of G. Constructing X takes $O(m)$ time, where m is the number of edges in G. Then for each vertex v of G, construct a set S_v to be the set containing all the edges incident to v; i,e., for each $v \in V$, construct

$$S_v = \{e \in E \mid e \text{ is incident to } v \text{ in } G\}.$$

Constructing S_v for all vertices v takes $O(n^2)$ time, where n is the number of vertices in G. Now let $F = \bigcup_{v \in V} \{S_v\}$. Intuitively, each vertex is associated with the set of edges it covers. Finally let $k' = k$. Output (X, F, k') as the set cover instance. The reduction runs in $O(n^2 + \log_2 k)$ time which is polynomial in the size of the vertex cover instance.

Correctness:

If G has a vertex cover V' of size k, then we claim that the corresponding sets S_v for each $v \in V'$ form a set cover for X. The reason is that by definition of a VC, every edge is incident to a vertex in V' implying that every $e \in X$ is a member of the set corresponding to a vertex in V'. This collection of sets forms a set cover of size $k = k'$. Conversely, if $C = \{S_{v_1}, S_{v_2}, \ldots, S_{v_k}\}$ is a set cover for X, then every $e \in X$ is in one of these sets, implying that the corresponding vertices $V' = \{v_1, v_2, \ldots, v_k\}$ form a vertex cover of size k for G.

6.

Set-Partition (PART) $\in$ NP

First we show that partition is in NP. The certificate for the partition problem consists of the two sublists S_1 and S_2. Given the sublists, in polynomial time we can compute the sums of the elements in these lists and verify that they are equal.

Subset Sum $\leq_P$ PART

Goal:

Next we want to show that subset sum (SS) is polynomially reducible to the set-partition problem (PART). That is, we want a polynomial time computable function f, which given an instance of SS (a set of numbers $S = \{x_1, x_2, \ldots, x_n\}$ and a target value t) outputs an instance of PART (a set of numbers $S' = \{x'_1, x'_2, \ldots, x'_n\}$) such that S has a subset summing to t if and only if S' can be partitioned into subsets S_1 and S_2 that sum to the same value.

Transformation:

Observe that the set-partition problem is a special case of the subset sum problem where we are trying to find a set of numbers S_1 that sum to half the total sum of the whole set. Let T be the sum of all the numbers in S. That is, $T = \sum_{i=1}^{n} x_i$. If $t = T/2$ then the subset sum problem is an instance of the partition problem, and we are done. If not, then the reduction will create a new number, which if added to any subset that sums to t, will now cause that set to sum to half the elements of the total. The problem is that when we add this new element, we change the total as well, so this must be done carefully.

We may assume that $t \leq T/2$, since otherwise the subset sum problem is equivalent to searching for a subset of size $T - t$, and then taking the complement. Create a new element $x_0 = T - 2t$, and call partition on this modified set. Let S' be S together with this new element: $S' = S \cup \{x_0\}$. Clearly the transformation can be done in polynomial time.

Correctness:

To see why this works, observe that the sum of elements in S' is $T+T-2t = 2(T-t)$. If there is a solution to the subset sum problem, then by adding in the element x_0 we get a collection of elements that sums to $t + (T - 2t) = T - t$, but this is one half of the total $2(T - t)$, and hence is a solution to the set-partition problem. Conversely, if there is a solution to this set-partition problem, then one of the halves of the partition contains the element x_0, and the remaining elements in this half of the partition must sum to $(T - t) - (T - 2t) = t$. Thus these elements (without x_0) form a solution to the subset sum problem.

Here is an example. $S = \{5, 6, 7, 9, 11, 13\}$ and $t = 21$ is an instance of the subset sum problem. $T = 51$. We create the element $x_0 = T - 2t = 9$ and add it to form $S' = \{5, 6, 7, 9, 9, 11, 13\}$. Note that there is a solution to the subset sum since $5 + 7 + 9 = 21$. The total of elements in S' is 60, and so by including x_0 we get a solution to the partition problem $5 + 7 + 9 + 9 = 30 = 60/2$.

7.