CMPSCI 311: Introduction to Algorithms **Spring 2018**

Homework 6 Solutions

Released 4/18/2018 Due 11:59pm 5/01/2018 in Gradescope

Instructions. You may work in groups, but you must individually write your solutions yourself. List your collaborators on your submission.

If you are asked to design an algorithm as part of a homework problem, please provide: (a) the pseudocode for the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm, and (e) justification for your running time analysis.

There are 4 questions, each worth 25 points total.

Submission instructions. This assignment is due by 11:59pm on 4/30/2018 in Gradescope. Please submit a pdf file. You may submit a scanned handwritten document, but a typed submission is preferred.

1. **Hitting Sets (K&T Ch8.Ex5)** For any H is possible to check where H is a hitting set of size at most k in polynomial time and so HittingSet $\in NP$. This can be done by iterating through all items in the proposed hitting set and checking-off the sets that they cover.

   We proved in class that VertexCover is NP-complete so it suffices to argue that VertexCover $\leq_P$ HittingSet and HittingSet $\in NP$. To prove VertexCover $\leq_P$ *HittingSet* we transform an instance (G, k) of VertexCover into a collection of sets: for each edge $e = (u, v)$ in G we generate the set $B_e = \{u, v\}$. The items are just the vertices $A = V$. Then observe that H is a hitting set of the collection of sets generated iff H is vertex cover in G.

   More formally, if there exists a hitting set of size k, then the corresponding vertices form a vertex cover of size k. This is apparent since each set B_e was hit, which means that for each edge, the set contains at least one of the end points. The converse is also true: if there exists a vertex cover, then the corresponding items form a hitting set.

   Clearly the reduction takes polynomial time.

2. **Path Selection (K&T Ch8.Ex9)**

   Given a set of k paths, we can ensure that there are no shared nodes in polynomial time. This is done be going through the list of all nodes of the paths (up to kV) and checking for duplicates. So PathSelection $\in NP$.

   We proved in class that IndependentSet is NP-complete so it suffices to argue that IndependentSet $\leq_P$ PathSelection and PathSelection $\in NP$. To show IndependentSet $\leq_P$ PathSelection, we transform an instance $G_I = (V_I, E_I)$ with parameter k of IndependentSet into a PathSelection problem instance also with parameter k. We construct the graph here as follows. Number the edge in E_I in some fashion, say $e_{I,1}, \ldots, e_{I,m}$. For each edge in the graph G_I, we will create a vertex in the new graph G_P, so for edge $e_{I,i}$ we will create a vertex $v_{P,i}$. For each vertex v_I in the original graph, we will form a path that is constructing by visiting its incident edges, in order. Formally, if v_I has incident edges $e_{I,j_1}, \ldots, e_{I,j_\ell}$ where $j_1 < \ldots < j_\ell$, we form the path $(v_{P,j_1}, v_{P,j_2}, \ldots, v_{P,j_\ell})$ in G_P. We do this for all of the vertices, and this new graph, along with these n paths forms the input to the PathSelection instance. Constructing this transformed problem takes $O(|V||E|)$ time.

   If we can find an independent set of size k in $G_I = (V_I, E_I)$ graph, then the k paths corresponding to each of the k vertices will not share any nodes in the graph G_P. This is by construction, path P_i only visits the edges incident to the corresponding vertex, so if v_i, v_j do not share an edge, the corresponding paths will not visit the same vertex in G_P. Since no edge exists between any of the k vertices, no edge will be common among their incident edges.

Similarly, if we can find k paths with no shared nodes, then we know that no edge exists between any of the k corresponding vertices.

Since this reduction can be done in polynomial time and produces correct results in the transformed problem, then IndependentSet $\leq_P$ PathSelection.

3. **Combinatorial Auctions (K&T Ch8.Ex13)** Given a set of bids and a bound B paths, we can ensure that there are no shared items and a total value of at least B in polynomial time. This is done be going through the list of all nodes of the paths (up to kV) and checking for duplicates. So Winner Determination $\in NP$.

   We proved in class that IndependentSet is NP-complete so it suffices to argue that IndependentSet $\leq_P$ Winner Determination and Winner Determination $\in NP$.

   To show IndependentSet $\leq_P$ Winner Determination, we transform a problem instance of IndependentSet with graph $G = (V, E)$ into a Winner Determination problem with V bids and E items. Create an item for each edge in G, and create a bid for each vertex in G. Let the bid corresponding to a vertex v consist of the items corresponding to the the edges incident to v. If the IndependentSet problem asks to find an independent set of size k, then set the value of each bid to 1 and look to collect an amount of k from the bids. Generating this transformed problem takes up to $O(|V||E|)$ time.

   If we can find an independent set of size k in $G = (V, E)$, then select the bids corresponding to each vertex in the independent set. Since no two vertices in the independent set share any edges, then no two corresponding bids will share any items. Then we are able to select k bids with no shared items, for a total dollar value of k which meets the requirement.

   Similarly, if we can collect a dollar value of at least k, then at least k bids can be accepted. No two of these bids share an item, which means the corresponding vertices do no share an edge. So picking the set of k corresponding vertices will form an independent set.

   Since this reduction can be done in polynomial time and produces correct results in the transformed problem, then IndependentSet $\leq_P$ WinnderDetermination.

4. **3-Coloring (K&T Ch13.Ex1)** Suppose we randomly assign a color to each node. This can be done in $O(n)$ time where n is the number of nodes. The probability that an edge (u, v) is satisfied is $6/9 = 2/3$ because there are 3×3 ways to color u and v and 3×2 ways to color the nodes such that the edge is satisfied. Hence, by linearity of expectation, the expected number of satisfied edges in the graph is $2m/3$ where m is the total number of edges in the graph. Since $c^* \leq m$, the expected number of satisfied edges is at least $2c^*/3$.