CS303 (Spring 2008)— Solutions to Assignment 13

Exercise 8.1-3

As we saw in class, the way to think about a comparison-based sorting algorithm is as a binary tree with decisions based on the outcomes of comparisons. Then, each leaf of the tree, having been reached by different paths, corresponds to a different initial ordering of the elements. Each such leaf must contain a different sequence of swap operations, as the same sequence of operations cannot possibly sort two arrays with initially different orderings. The total number of orderings of n elements is $n!$, and the total number of leaves of a binary tree of maximum height h is at most 2^h (less if the tree is unbalanced). Because there were $n!$ leaves, this meant that $2^h \geq n!$, so $h \geq \log_2 n! = \Omega(n \log n)$.

How about half the inputs? That way, we are still looking at at least $\frac{1}{2}n!$ leaves, and the tree is still binary, so we now have to solve $2^h \geq \frac{1}{2}n!$, or $2^{h+1} \geq n!$. Thus, we get $h+1 \geq \log_2 n!$, and still $h = \Omega(n \log n)$.

Moving on to a fraction of $1/n$, we consider $1/n \cdot n! = (n-1)!$ leaves, so the necessary height of the tree is $h \geq \log_2(n-1)! = \Omega((n-1)\log(n-1)) = \Omega(n \log n)$, so asymptotically, we again do no better.

Even with only a fraction of $1/2^n$, there are still $n!/2^n$ leaves whose height matters, and we get a necessary height of the tree containing them as

$$h \quad \geq \quad \log_2(n!/2^n) \quad = \quad \log_2(n!) - \log_2(2^n) \quad = \quad \Omega(n \log n) - n \quad = \quad \Omega(n \log n).$$

So even if we only care about a $1/2^n$ fraction of inputs, the worst case over all those inputs is still $\Omega(n \log n)$ for any comparison-based sorting algorithm.

Exercise 34.5-1

Notice that this problem is different from Graph Isomorphism: in the latter, you are given two graphs G_1, G_2 (presumably of the same size), and are to decide if the two are isomorphic. Graph Isomorphism is rather famous for the fact that neither is it known to be NP-complete (most researchers believe that is rather unlikely), nor does anyone have a polynomial-time algorithm. Subgraph Isomorphism, on the other hand, is known to be NP-complete, and we will prove that here.

To see that the problem is in NP, we observe that a certificate is a mapping ϕ from the nodes of G_1 to (a subset of) the nodes of G_2, describing which vertices of G_2 correspond to vertices of G_1. The certifier then needs to make sure that for each edge $e = (u, v)$ in G_1, the edge $(\phi(u), \phi(v))$ is also in G_2, and whenever (u, v) is not an edge of G_1, then $(\phi(u), \phi(v))$ is not an edge of G_2. This can be done with two simple nested loops, and takes at most $O(n^2)$ time, i.e., polynomial.

To prove NP-hardness, we show that, for instance, the Clique problem is a special case. Given an instance of Clique, consisting of a graph G and a number k, we generate an instance of Subgraph Isomorphism by setting $G_2 = G$, and G_1 a clique on k vertices. This reduction clearly takes polynomial time, since all we do is copy a graph, and write down a complete graph in time $O(k^2)$.

To prove correctness of the reduction, first assume that G contains a clique of size k. Then, those k nodes of $G = G_2$ are isomorphic to G_1, because G_1 is a clique of size k.

Conversely, if G_2 contains a subgraph isomorphic to G_1, because G_1 is a k-clique, G_2 must contain a k-clique. Thus, (G, k) is a "Yes" instance.

Set Cover

The decision version is as follows: Given $U, S_1, \ldots, S_m$, and a target number k, is there a set $C \subseteq \{1, \ldots, m\}$ of at most k indices with $\bigcup_{i \in C} S_i = U$?

To see that the problem is in NP, we give an efficient certifier. The certificate is the set C. We count its size to make sure it contains at most k indices, and then use, say, a bit vector to test that each element of U is covered by at least one S_i with $i \in C$. This clearly takes polynomial time.

To prove NP-hardness, we reduce from VERTEX COVER. The input to VERTEX COVER is a graph G and a target number k, and we have to produce the universe U and the sets $S_1, S_2, \ldots$, as well as a target number k'. The intuition is that in VERTEX COVER, each vertex covers a set of edges. So we will have an element u_e for each edge e, and a set for each vertex. Formally, we will have $U = E(G)$, and for each vertex $v_1, v_2, \ldots, v_i, \ldots, v_n$, we have a set $S_i = \{e \in E \mid v_i \text{ is incident on } e\}$. That is, S_i is exactly the set of edges that v_i covers. Finally, we set $k' = k$. This reduction is really mostly a renaming, and clearly runs in polynomial time.

To verify that it is correct, first assume that G has a vertex cover of size at most k. Say that it consists of the vertex set T. Now, we construct C by including exactly the S_i with $v_i \in T$. Clearly, the size of C is equal to the size of T, i.e., at most $k = k'$. Each element u_e of U corresponds exactly to an edge e in E. Because T is a vertex cover, e is covered by some vertex $v_i \in S$. But that means that $u_e \in S_i$ by definition of the sets S_i, so u_e is also covered by C. Since this holds for each u_e, C is a valid set cover.

For the opposite direction, assume that the new instance has a set cover C of size at most $k' = k$. We define a vertex set T as consisting exactly of those v_i for which $i \in C$. The size of T is the same as that of C, i.e., at most k. Furthermore, for any edge e, we know the corresponding element u_e is covered by at least one set S_i with $i \in C$. Therefore, by the way we generated the S_i sets, the vertex v_i covers e. Since all edges are thus covered, T is a vertex cover of size k, proving that the instance was a "Yes" instance.

Set Packing

The decision version is as follows: Given $S_1, \ldots, S_m$, and a target number k, is there a set $C \subseteq \{1, \ldots, m\}$ of at least k indices with $S_i \cap S_j = \emptyset$ for all $i, j \in C, i \neq j$?

To see that the problem is in NP, we give an efficient certifier. The certificate is the set C. We count its size to make sure it contains at least k indices, and then try each pair of sets S_i, S_j with $i, j \in C$ to see if they intersect. That takes time at most $O(k^2 m)$, where m is the size of the largest set, and is thus polynomial.

To prove NP-hardness, we use a nearly identical reduction to the one for the previous problem, except this time from INDEPENDENT SET. The input to INDEPENDENT SET is a graph G and a target number k, and we have to produce the sets $S_1, S_2, \ldots$, as well as a target number k'. The intuition is that in INDEPENDENT SET, each vertex "comes with" the set of all its incident edges, and if two vertices have an incident edge in common, they cannot be picked simultaneously. Having an incident edge in common means that the sets of incident edges intersect, so we generate, for each vertex v_i, a set S_i containing exactly the edges e incident on v_i. Finally, we again set $k' = k$. This reduction clearly runs in polynomial time, since again, it is mostly a renaming.

To verify that it is correct, first assume that G has an independent set of size at least k, call it T. Construct C by including exactly the S_i with $v_i \in T$ (as before). The size of C is equal to the size of T, i.e., at most $k = k'$. Furthermore, since no two vertices in C can share an edge, the sets S_i we picked must be pairwise disjoint, so we have found a valid set packing of at least k sets.

For the opposite direction, assume that the new instance has a set packing C of size at least $k' = k$. Again, we define a vertex set T as consisting exactly of those v_i for which $i \in C$. The size of T is the same as that of C, i.e., at most k. For any edge e, at most one set S_i with $i \in C$ contains e, so at most one node v_i can be incident on e. Thus, no two selected nodes are connected by an edge, and T is in fact an independent set of size at least k, proving that (G, k) was a "Yes" instance.