Advanced Analysis of Algorithms - Homework III (Solutions)

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. Is there a problem in the complexity class P, such that all problems in P can be polynomially transformed to this problem?

   Solution: Yes; for instance consider the language $L = \{5\}$, i.e., the language containing a single element and the corresponding decision problem: D_1: Does $x \in L$?. I can reduce an arbitrary decision problem $D_2 \in \mathsf{P}$ to D_1 using the reduction f, which is defined as follows: *Given x as input to D_2, f first checks in polynomial time, whether $x \in D_1$. If x is a "yes" instance of D_2, $f(x) = 5$; otherwise $f(x) = 6$. It is not hard to see that an instance x is a "yes" instance of D_2 if and only if $f(x)$ is a "yes" instance of D_1!* In other words, we have reduced D_2 to D_1. □

2. Show that a language L can be *verified* in deterministic polynomial time if and only if it can be decided by a non-deterministic algorithm in polynomial time.

   Solution:

   If: Assume that a language L can be decided by a non-deterministic algorithm in polynomial time.

   This means that there exists a non-deterministic Turing Machine N, which decides whether a given $x \in L$, in time $O(p(|x|))$, where $p(n)$ is a fixed polynomial. As described in class, the computation of N on x is a tree T, with each branching representing a non-deterministic choice. Each path from the root to an accepting leaf of this computation tree constitutes a proof. Since the non-deterministic Turing Machine takes polynomial time, the depth of the computation tree is $O(p(|x|))$, on an input x; further the time spent at each node is also bounded by a polynomial, say $q(|x|)$. It follows that there exists a verification algorithm for L that runs in time $O(p(|x|) \cdot q(|x|))$, i.e., in time polynomial in the size of the input.

   Only If: Assume that L can be verified in deterministic polynomial time. This means that given the query: Does $x \in L$, there exists a proof $Y(x)$ if $x \in L$, such that $|Y(x)| \leq q(|x|)$ and that $Y(x)$ can be checked for correctness in time $p(|x|)$, where $p(n)$ and $q(n)$ are fixed polynomial functions. Now consider the following non-deterministic algorithm to decide L: Given an input x, first guess $Y(x)$ and then verify that $Y(x)$ is a valid proof for $x \in L$. The running time of this algorithm is $q(|x|) + p(|x|)$, which is clearly polynomial. □

3. Design a backtracking algorithm for the 3SAT problem.

   Solution: Let us say that we are given a 3CNF formula $\phi = C_1 \wedge C_2 \ldots \wedge C_m$, where each clause C_i is a disjunction of three literals on the variables $V = \{x_1, x_2, \ldots x_n\}$. The crucial check is whether or not the current (partial) assignment can be completed to a full assignment. Start with $x_1 = \mathbf{true}$ and use this assignment to get a reduced set of clauses; the clauses in which x_1 occurs in uncomplemented form can be deleted from the clause set and the clauses in which x_1 occurs in complemented form have one literal less. Call this formula ϕ_1^t. Recursively extend this assignment if possible, by setting $x_2 = \mathbf{true}$ and so on; at each node of the tree, a check can be made as to whether the current assignment can be extended. If not, close that path and proceed to the parent node which sets the current variable to **false**. In this manner the search space is explored using backtracking. □

4. Consider an instance of the Subset-Sum problem, where $S = \{2, 10, 13, 17, 22, 42\}$ and $B = 52$. Solve this instance using backtracking, showing all the steps.

   Solution: This is too painful for me! The answer is yes, with subset $S' = \{10, 42\}$. This solution is obtained by excluding the element 2 and backtracking. □

5. Consider the following graph coloring algorithm for coloring the vertices of a graph using the fewest number of colors:

Function FIND-OPTIMAL-COLOR(**G=**$<$**V,E**$>$)

Let $V_{un} = V$ and $C_u = \{1, 2, \ldots, n\}$.
while ($V_{un} \neq \phi$) **do**
  c_{cur} is the smallest indexed color in C.
  Assign c_{cur} to as many vertices as possible in V_{un} making sure that a vertex with index number k is considered before a vertex with index number $k + 1$.
  Delete all the colored vertices from V_{un}.
  Delete c_{cur} from C.
end while

Algorithm 1.1: Graph Coloring Algorithm

V_{un} is the set of uncolored vertices and C_u is the set of unasssigned colors.

Is Algorithm (1.1) optimal? Justify your answer with a proof or a counterexample.

Solution: The algorithm is clearly suboptimal; for instance, consider the following graph:

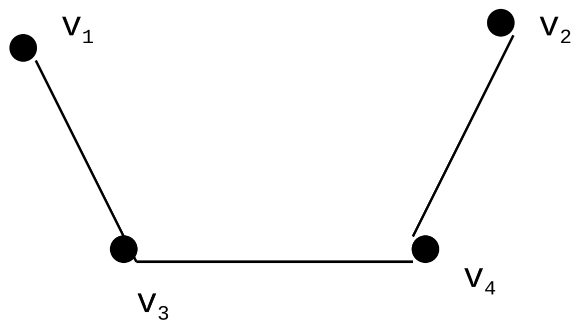

Figure 1: Counterexample to Algorithm (1.1)

It is clear that Algorithm (1.1) will require 3 colors, whereas 2 colors are sufficient. □