Illinois Institute of Technology
Department of Computer Science

Homework 8 Solutions

CS 430 Introduction to Algorithms
Fall Semester, 2010

1. **Problem 34.4-6 on page 1086**.

   Solution: Suppose that we have a polynomial time algorithm IS-SATISFIABLE to decide if a given formula with variables $x_1, x_2, \ldots, x_n$ is satisfiable. We can use this to find a satisfying assignment in polynomial time. First use IS-SATISFIABLE to determine if the formula is satisfiable. If it is not, stop now since there is no point in trying to find an assignment. However, if the formula is satisfiable, replace each occurrence of the variable x_1 in our formula with the value *true*, then use IS-SATISFIABLE to decide if this formula is satisfiable. If it is, then $x_1 = true$, otherwise, $x_1 = false$. Now, keeping our assignment for x_1, set $x_2 = true$ in our formula and check if it is satisfiable. If so, then $x_2 = true$, otherwise $x_2 = false$. Repeat this process for each x_i, each time recording the satisfying truth assignment. This approach takes polynomial time since IS-SATISFIABLE polynomial time and we call IS-SATISFIABLE $n+1$ times (once with the original formula and once for each of the n variables in our formula).

2. **Problem 34.4-7 on page 1086**.

   Solution: We can solve a 2-CNF-SAT problem in polynomial time by reducing it (in polynomial time) to the problem of finding strongly connected components in a directed graph. Suppose that we are given a 2-CNF-SAT problem with variables $x_1, x_2, \ldots, x_n$. We may assume that each clause in our formula has two literals (any single literal clauses could be replaced with the clause $(x \vee x)$).

   Using the hint, we observe that each clause $(x \vee y)$ can be converted into two equivalent implications, $\neg x \rightarrow y$ and $\neg y \rightarrow x$. We use these implications to construct a directed graph $G = (V, E)$. This graph is created as follows.

   - $V = \{x_1, x_2, \ldots, x_n, \neg x_1, \neg x_2, \ldots, \neg x_n\}$
   - For each clause, convert it to its two equivalent implications and for each implication $x \rightarrow y$ add the directed edge (x, y) to the graph. For example, the clause $(x_i \vee \neg x_j)$ is equivalent to the implications $\neg x_i \rightarrow \neg x_j$ and $x_j \rightarrow x_i$. As a result we add the directed edges $(\neg x_i, \neg x_j)$ and (x_j, x_i) to E.

   Now we show that a 2-CNF-SAT formula is satisfiable if and only if there is no strongly connected component (SCC) in G that contains both a variable and its complement (i.e. no SCC contains both x_i and $\neg x_i$ for some i).

   Since paths in the graph G represent a chain of implications, if two literals x and y are in the same SCC, then we have $x \rightarrow y$ and $y \rightarrow x$. The only way to satisfy both implications is if $x = y$. If x and y complements (i.e. $x = x_i$ and $y = \neg x_i$), it is impossible for them to have the same truth value, therefore the formula is not satisfiable if they are in the same SCC.

   Now suppose that none of our SCCs contain both a variable and its complement. Construct a component graph G^{SCC} by contracting all edges within SCCs of G (see Figure 22.9 on page 616 for an example). In G^{SCC} each SCC in G is replaced with single vertex. All edges between vertices in different SCCs in G are now edges between their component vertices in G^{SCC}. Now perform a topological sort of G^{SCC}. For each variable x_i, if its component vertex in G^{SCC} comes before the component vertex of

its complement $\neg x_i$ in the topological sort, then set $x_i = false$, otherwise set $x_i = true$. Clearly this approach takes polynomial time (linear time, technically) since constructing the graph, computing the SCCs and performing a topological sort of the component graph all take polynomial time. To show that this approach works we need to show that it assigns the same truth value to all literals in the same SCC and that if there is a path from SCC A to SCC B then we will not have literals in A assigned the value *true* while those in B are assigned the value *false*.

First we show that all literals in a SCC are assigned the same truth value. We shall prove this by contradiction. Suppose that we have the SCCs A, B, and C and they appear in this order in the topological sort. Now suppose that we have the literals x and y which are both in B and that $\neg x \in A$, but $\neg y \in C$. Hence, $x = true$ and $y = false$. Since $x, y \in B$, we have $x \rightarrow y$ and $y \rightarrow x$. Since for each edge (x_i, x_j) that we add to G, we also add the edge $(\neg x_j, \neg x_i)$, if the graph has a path from x to y, then it must also have a path from $\neg y$ to $\neg x$. However, we cannot have such a path since we assume that $\neg x \in A$ and $\neg y \in C$ and that C comes after A in the topological sort.

Now we use contradiction to show that it truth values between SCCs will satisfy our formula. Suppose that we have SCCs A and B and that there is a path from A to B. Furthermore, assume that literals in A are assigned the value *true* and those in B are assigned *false*. Let a and b be literals such that $a \in A$ and $b \in B$. Note that a and b cannot be complements of each other (i.e. we cannot have $a = x_i$ and $b = \neg x_i$). Otherwise $b = true$ and $a = false$ since A appears before B in the topological sort. Since G contains a path from a to b, it must contain a path from $\neg b$ to $\neg a$, let $\neg b \in \neg B$ and $\neg a \in \neg A$. Since $b = false$ and $a = true$, then $\neg B$ is assigned *true* and $\neg A$ is assigned *false*. As a result, our topological order must have B coming before $\neg B$ and $\neg A$ coming before A. But this gives us a cycle unless $A, B, \neg A, \neg B$ are all in the same SCC. But this contradicts our assumption that A and B are distinct SCCs.

3. **Problem 34.5-7 on page 1101**.

   Solution: The longest-simple-cycle of a graph is a simple cycle that contains the most edges of any simple cycle in the graph and does not visit any vertex more than once. The related decision problem asks if a graph contains a simple-cycle with at least k edges. To prove that the decision problem is NP-complete, we need to show that a solution can be verified in polynomial time and that the decision problem is at least as hard as any other NP-complete problem.

   A solution to the decision problem is a simple cycle with at least k edges. To verify the solution, we need to check each edge to make sure that a vertex is not visited more than once and that the number of edges is at least k. This can easily be done in polynomial time in terms of the number of edges and vertices in the graph.

   To prove that the longest-simple-cycle decision (LSCD) problem is NP-hard we show that the Hamiltonian cycle (HAM-CYCLE) problem $\leq_p$ LSCD. Let $G = (V, E)$ be an instance of HAM-CYCLE. We construct an instance of LSCD as follows, $G' = (V, E)$ and $k = |V|$. Clearly this reduction takes polynomial time as G' is the same graph as G.

   Now we show that G contains a Hamiltonian cycle if and only if G' contains a simple cycle of at least k edges, where $k = |V|$. First note that a simple cycle with k edges visits k different vertices (a spanning tree of k vertices has $k - 1$ edges, add one more and you get a cycle that visits k vertices). If G contains a Hamiltonian cycle, then the edges in this cycle are also a simple cycle of length k of G', since a Hamiltonian cycle is a simple cycle (no repeated vertices) and has $|V|$ edges. If G' has a simple cycle of length k, the edges in this cycle are a Hamiltonian cycle of G since the cycle visits $k = |V|$ vertices without repeating any vertices.