BU CLA CS 530: Analysis of Algorithms
Spring 1999

Selected Solutions for Problem Set 11

Problem 1. (Exercise 36.4-7, page 946, in [CLR])

We follow the hint given in [CLR]. Note that the book says that every clause has "exactly 2 literals", but this restriction can be relaxed to "at most 2 literals" because every 1-literal clause (l) can be replaced by the 2-literals clause $(l \vee l)$. Think of every clause $(l \vee l')$ as a pair of implications

$$(\neg l \rightarrow l') \quad \text{and} \quad (\neg l' \rightarrow l)$$

Construct a directed graph $G = (V, E)$ with a vertex for every literal and directed edges corresponding to the implications above. If the given formula φ has n clauses, G has at most $4n$ vertices and $2n$ edges. We can therefore construct G from φ in linear time.

We claim that φ is satisfiable iff no pairs of complementary literals (i.e. x and $\neg x$) appear in the same strongly connected component of G. Under any satisfying truth assignment, all literals in a strongly connected component of G must have the same truth value. Hence, if a variable x appears both positively (as x) and negatively (as $\neg x$) in the same strongly connected component of G, then φ is not satisfiable.

Conversely, suppose that no pair of complementary literals appear in the same strongly connected component of G. Construct a new directed graph $G' = (V', E')$ by collapsing each of the strongly directed components of G into a single vertex. (The construction of G' is not part of the algorithm, and only used to prove the correctness of our algorithm.) Formally, the vertices of G' are:

$$V' = \{ [v] \mid v \in V \} \qquad \text{where} \quad [v] = \{ w \mid v \equiv w \}$$

and where: $v \equiv w$ iff "v and w are in the same strongly connected component". What we have done is to define an equivalence relation $\equiv$ on V, then take V' as V modulo $\equiv$, sometimes written as $V' = (V/\equiv)$. The edges of G' are:

$$E' = \{ ([v], [w]) \mid (v, w) \in E \}$$

G' is acyclic, because we have collapsed every strongly connected component into a single vertex. Hence, its vertices can be topologically sorted (see Section 23.4 in [CLR]), i.e. they can be totally ordered. We define a truth assignment satisfying the given formula φ as follows: Assign value FALSE (or 0) to variable x if the strongly connected component of x occurs before the strongly connected component of $\neg x$ in this total ordering, and assign value TRUE (or 1) to x if the strongly connected component of $\neg x$ occurs before the strongly connected component of x. It is straightforward to check this truth assignment indeed satisfies φ.

We can find the strongly connected component of G in linear time (Section 23.5 in [CLR]). Hence we have a linear-time algorithm for 2-CNF-SAT.

Problem 2. (Exercise 36.5-4, page 960, in [CLR])

We omit the easy argument that PARTITION is in NP. The non-trivial part is to show PARTITION is NP-hard. This is shown by the reduction SUBSET-SUM $\leq_{\mathrm{P}}$ PARTITION. Let $\langle S, t\rangle$ be an instance of SUBSET-SUM, where $S = \{a_1, \dots, a_n\}$ and $t, a_1, \dots, a_n$ are natural numbers.[1] Let $s = a_1 + a_2 + \cdots + a_n$ and introduce two new numbers a_{n+1} and a_{n+2}, defined by:

$$a_{n+1} \;=\; 1 + t \qquad \text{and} \qquad a_{n+2} \;=\; 1 + s - t$$

(We can assume $t \leq s$, otherwise we can immediately return *no*, there is no solution for this instance of SUBSET-SUM.) We have chosen a_{n+1} and a_{n+2} so that they cannot go into the same set of a two-set partition, because together they outweigh all the other elements, i.e. $a_{n+1} + a_{n+2} = 2 + s$. The desired input instance for PARTITION is $S' = S \cup \{a_{n+1}, a_{n+2}\}$, which is constructed in time polynomial in the size of $\langle S, t\rangle$ (why?). We now ask if this new set S' can be partitioned into two subsets X' and Y' such that $\sum X' = \sum Y'$. Equivalently, by our choice of a_{n+1} and a_{n+2}, we ask if S can be partitioned into two subsets X and Y such that $a_{n+1} + \sum X = a_{n+2} + \sum Y$. Then

$$a_{n+1} + \sum X = a_{n+2} + \sum Y \quad\Longleftrightarrow\quad 1 + t + \sum X = 1 + s - t + \sum Y \quad\Longleftrightarrow\quad \sum Y = t$$

where the last double implication follows from the fact that $s = (\sum X) + (\sum Y)$.

Related Exercise (not part of the homework but a good review for the end-of-term test): Show that PARTITION $\leq_{\mathrm{P}}$ SUBSET-SUM. For this, an instance $S = \{a_1, a_2, \dots, a_n\}$ of PARTITION is transformed to an instance of SUBSET-SUM, $\langle S, t\rangle$ where $t = (1/2)(a_1 + a_2 + \cdots + a_n)$. The transformation from S to $\langle S, t\rangle$ is polynomial in the size of S (why?), and the answer to the SUBSET-SUM problem is *yes* (or *no*) iff the answer to the PARTITION problem is *yes* (or *no*).

Problem 3. (Exercise 36.5-6, page 960, in [CLR])

We show that **ham-cycle** is polynomial time reducible to **longest-simple-cycle**. This will show that **longest-simple-cycle** is NP-hard, because **ham-cycle** is NP-complete. An algorithm to solve **ham-cycle** using as a subroutine an algorithm for **longest-simple-cycle** is:

HAM-CYCLE (G)
1. $p \leftarrow$ LONGEST-SIMPLE-CYCLE (G)
2. $k \leftarrow$ number of vertices in cycle p
3. **if** $k < |V|$
4. **then return** "G does not have a hamiltonian cycle"
5. **else return** "G has a hamiltonian cycle"

Assaf Kfoury
Created: 99.04.08

[1] Instances of SUBSET-SUM and PARTITION in the literature are usually restricted to natural numbers, or even positive integers. [CLR] makes such a restriction for instances of SUBSET-SUM only (page 951), and allows instances of PARTITION to be sets of numbers in general (page 960). Our solution here makes the restriction for both problems. The NP-hardness of PARTITION with the restriction, which requires some extra care, implies its NP-hardness without the restriction.