Solution

Computational Complexity – Homework 3

Discussed on 02.05.2016.

Exercise 3.1

Is **NP** closed under intersection, resp. union?

Exercise 3.2

Prove that DOUBLE-SAT $= \{\langle\Phi\rangle \mid \Phi$ is a Boolean formula with at least two satisfying assignments $\}$ is **NP**-complete.

Exercise 3.3

(a) Let M be a Turing machine which decides SAT, and let ϕ be a CNF formula with n variables.

Design a recursive algorithm which computes a satisfying assignment for ϕ (if ϕ is satisfiable) using at most $2n+1$ calls to M plus some additional polynomial-time computation.

(b) Assume that $L \subseteq \{1\}^*$ is a *unary* language which is also **NP**-complete.

Show that then SAT $\in$ **P**.

Hints:

- Again write a recursive program but limit the number of recursive calls by using a hash map. Use as hash function a polynomial-time reduction f of SAT to L.
- Consider then the call tree of your program for a given input. Show that two nodes v, v' which do not lie on a common path from the root to a leaf correspond to formulae $\phi_v, \phi_{v'}$ with $f(\phi_v) \neq f(\phi_{v'})$.

Solution:

(a) Let $x_1, \ldots, x_n$ be the variables of ϕ. We recursively calculate a satisfying assignment as follows:

(b) Let L be the unary **NP**-complete language. Then SAT is reducible in polynomial time to L, i.e., there is a function f such that for every CNF ϕ we have

$$\phi \in \text{SAT} \Leftrightarrow 1^{f(\phi)} \in L.$$

We use this f as a hash function in order to limit the number of recursive calls. For this, note that we further have a polynomial p such that $p(|\phi|)$ is the time needed to compute $1^{f(\phi)}$. Hence, $f(\phi) \leq p(|\phi|)$.

Consider the call tree $T = (V, E)$ of satisfiable for an input formula, i.e., every node $v \in V$ corresponds to an instance of satisfiable , every edge corresponds to a recursive call of one instance by another. For $v \in V$ let ϕ_v be the formula the instance v has as argument.

Consider now two nodes v, v' such that neither one is an ancestor of the other, i.e., there is no path from the root to a leaf which visits both nodes. Then wlog. the computation of v has already terminated when the computation of v' starts. So, at the time of the call of v' it is already known whether ϕ_v is satisfiable and, thus, the hashmap is defined for $f(\phi_v)$. Hence, $f(\phi_v) \neq f(\phi_{v'})$.

In contraposition, $f(\phi_v) = f(\phi_{v'})$ implies that v and v' are located on a common path from the root to some leaf. Every such path has length at most n, i.e., there are at most n nodes whose formula maps to the same hash value.

As $f(\phi_v) \leq p(|\phi|)$ for all $v \in V$, there are at most $n \cdot p(|\phi|) \leq |\phi| \cdot p(|\phi|)$ nodes.

Exercise 3.4

In the lecture, you have seen the definition of "polynomial-time reducible" $\leq_p$:

> For two languages $A, B \subseteq \{0,1\}^*$ we write $A \leq_p B$ if there is a function $f : \{0,1\}^* \to \{0,1\}^*$ computable in polynomial time such that $x \in A \Leftrightarrow f(x) \in B$ for all $x \in \{0,1\}^*$.

Similarly, the notion of "log-space reducible" $\leq_{\log}$ is defined but this time the function f has to be computable by a Turing machine using at most $\mathcal{O}(\log n)$ space.

(a) Show that $A \leq_{\log} B$ implies $A \leq_p B$.

(b) Show that for any two languages A, B in **P** *with* $B \neq \emptyset, \{0,1\}^*$ we have $A \leq_p B$.

Remark: Using $\leq_{\log}$ one can also define **P**-complete problems in a meaningful way.

(c) Argue that $\leq_{\log}$ is also transitive, i.e., if $A \leq_{\log} B \leq_{\log} C$, then also $A \leq_{\log} C$.

Hint: This is not as straightforward as for polynomial-time reductions. Why?

Solution:

(a) As **L** is contained in **P**, every function computable by a log-space TM is also computable by a poly-time TM.

More precisely: If M is a TM running in space $\mathcal{O}(\log n)$, then the number of possible configurations is at most exponential in the space used by the computation, i.e., $\mathcal{O}(2^{c \log n}) = \mathcal{O}(n^c)$ for some $c > 0$. As every computation visits any possible configuration at most once, the running time is polynomial in the input size.

(b) We assume $B \neq \emptyset, \{0,1\}^*$, otherwise the result does not hold in general.

The reduction is as follows:

Choose any $y \in B$ and any $z \notin B$. We then check in polynomial time if a given input $x \in A$. If $x \in A$, the reduction outputs y, otherwise z. Note that writing y or z takes constant time!

(c) We construct a TM M which basically behaves just like M_g, but everytime M_g needs to read the i-th bit of its input, i.e., the i-th bit of the output of M_f, M simply simulates M_f on input x (without storing its output!) until M_f writes the i-th bit (see Ex. 2.2(c)). As M_f only needs $\mathcal{O}(\log |x|)$ space, M can always simulate M_f.

Exercise 3.5

(a) Show that **NP**=co**NP** if and only if 3SAT and TAUTOLOGY are polynomial-time reducible to each other.

(b) A *strong* nondeterministic Turing machine (sNDTM) is a NDTM which has three possible outputs: "1", "0", "?". An sNDTM M decides a language L if: (i) for $x \in L$ every computation of M on x yields "1" or "?" and there is at least one computation of M on x which yields "1". (ii) for $x \notin L$ every computation of M on x yields "0" or "?" and there is at least one computation of M on x which yields "0".

Show that L is decided by an sNDTM in polynomaial time iff $L \in \mathbf{NP} \cap \mathrm{co}\mathbf{NP}$.

Exercise 3.6

Notation: For n a natural number let $[n]$ be the set $\{1, 2, \ldots, n\}$.

The KNAPSACK problem is defined as follows:

We are given n items where item i has both a weight $w_i \in$ and a value v_i. We are also given a maximal weight W the knapsack can hold and a target value V. (All numbers are assumed to be positive integers.) A selection $S \subseteq [n]$ then has total weight $w(S) := \sum_{i \in S} w_i$ and total value $v(S) := \sum_{i \in S} v_i$. A selection S is a *solution* if $w(S) \leq W$ and $v(S) \geq S$ hold.

(a) Give a reasonable encoding of KNAPSACK and show that KNAPSACK is in **NP**.

(b) Assume you are given an algorithm for deciding KNAPSACK running in polynomial time.

Construct from it a polynomial-time algorithm which computes the maximal $V_{\max}$ for which a given instance of KNAPSACK has a solution.

(c) Give an algorithm for deciding KNAPSACK in time $\mathcal{O}(nW)$.

Hint: Use dynamic programming to produce a table $V(w, i)$ where

$$V(w, i) := \max \{v(J) \mid J \subseteq [i] \text{ and } w(J) = w\}.$$

Remark: Note that W is exponential in the size of the representation of W.

(d) We define MULTI-KNAPSACK to be the problem where for every item $i \in [n]$ we are given M values v_i^p ($p \in [M]$) and N weights w_i^q ($q \in [N]$) with corresponding target values V^p and total weights W^q. (All numbers are assumed to be positive integers.) A selection $S \subseteq [n]$ is then a solution of the MULTI-KNAPSACK instance if

$$\forall p \in [M] : \sum_{i \in S} v_i^p \geq V^p \text{ and } \forall q \in [N] : \sum_{i \in S} w_i^q \leq W^q.$$

Show that MULTI-KNAPSACK is also in **NP** and give a reduction 3SAT $\leq_p$ MULTI-KNAPSACK .

Hint: The reduction is quite similar to 3SAT $\leq_p$ 0/1-IPROG: Given a 3CNF formula ϕ with M clauses and N variables, generate a MULTI-KNAPSACK instance with $n = 2N$ items, i.e., one for every literal, and $v_i^p, w_i^q \in \{0,1\}$ for $i \in [n], p \in [M+N], q \in [N]$. An truth assignment of ϕ should correspond to the selection of those literals which evaluate to true.

(e) Give a reduction 3SAT $\leq_p$ KNAPSACK .

Hint: Start from your reduction of 3SAT to MULTI-KNAPSACK and set $w_i := v_i := v_i^1 \dots v_i^{M+N}$ for $i \in [2N]$ and $W := V := 1^N 3^M$ with all strings interpreted as numbers in *decimal* representation. A satisfying assignment should then yield a selection of total weight/value in $[1^N 1^M, 1^N 3^M]$. Introduce $2M$ additional items which allow to extend every selection induced by a satisfying assignment to a solution of the KNAPSACK instance.

Solution:

(a) We may assume that an instance of KNAPSACK is given as a list of pairs v_i, w_i plus V, W, e.g.,

$$v_1, w_1, v_2, w_2, \dots, v_n, w_n \# V, W$$

(We use an input alphabet different from $\{0,1\}$ here.)

Then an NTM can simply scan the input once and decide nondeterministically for every $i \leq n$ whether i to include i in S or not. If $S := S \cup \{i\}$, then the NTM simply adds v_i, resp. w_i to the current total value, resp. total weight of S (stored on two separate work tapes). Finally it compares the total value, resp. weight to V, resp. W. All these steps can be done in time polynomial in the length of the input.

(b) Set $V_{\max} = \sum_{i=1}^n v_i$. Then use binary search on the intervall $[0, V_{\max}]$, i.e., first decide whether the given instance of KNAPSACK is solvable for $V := V_{\max}/2$. If it is, test if it solvable for $3/4 V_{\max}$; otherwise test if it is solvable for $V := V_{\max}/4$ and so forth.

Note that the binary representation of $V_{\max}$ is polynomial in the size of the input, so the number of considered KNAPSACK instances (at most $\log_2 V_{\max}$) is also polynomial in the size of the input.

(c) Obviously, $V(w, 0) = 0$ for all $w \leq W$. ($\sum_{i \in \emptyset} v_i = 0$.) Assume that $V(w, i-1)$ is known and corresponds to some selection $S \subseteq \{1, 2, \dots, i-1\}$. We then may consider including i into S, leading to the total weight $w + w_i$ and total value $V(w, i-1) + v_i$. Hence, $V(w + w_i, i) \geq v_i + V(w, i-1)$. This gives us the following algorithm:

(d) MULTI-KNAPSACK $\in$ **NP**:

The NTM nondeterministically chooses a selection S and stores the corresponding weights and values on a work tape. Then it checks the $N + M$ inequalities within $N + M$ iterations.

3SAT $\leq_p$ MULTI-KNAPSACK :

Consider a 3CNF formula ϕ with M clauses and N variables $x_1, \dots, x_n$.

We associate the items $1, \dots, N$ with the literals $x_1, \dots, x_n$, the items $N+1, \dots, 2N$ with the literals $\neg x_1, \neg x_2, \dots, \neg x_n$. A truth assigment of ϕ will correspond to the selection which contains exactly those literals which evaluate to true under the given assignment.

We define the weights and values for every literal:

For $p \in [N]$ set $v_i^p = w_i^p = 1$ if the corresponding literal is associated with variables x_p, otherwise $v_i^p = w_i^p = 0$.

For $p \in [M]$ set $v_i^{N+p} = 1$ if the literal corresponding to i appears in clause p; otherwise $v_i^{N+p} = 0$.

Every solution of the MULTI-KNAPSACK instance should also correspond to a satisfying assignment of ϕ. Hence, a solution S should never select both literals of a given variable x_i. We therefore set $W^i := 1$. Then $\sum_{k \in S} w_k^i = w_i^i + w_{i+N}^i \leq 1$ guarantees that S contains at most one of two literals.

Similarly, every solution S should contain at least one of the two literals of the variable x_i. So, we also set $V^i := 1$ for $i \in [N]$. Then $\sum_{k \in S} v_k^i = v_i^i + v_{i+N}^i \geq 1$ guarantees that S contains at least one literals of every variables.

As $v_k^i = w_k^i$ for $i \in [N]$ every solution S selects exactly one literal for every variable and defines, thus, an assignment for ϕ.

Finally, for every clause a solution S should contain at least one literal. So we set $V^{N+i} := 1$ for $i \in [M]$. Then

$$\sum_{k \in S} v_k^{N+i} = \sum_{\text{Literal } k \text{ appears in clause } i} v_k^{N+i} \geq 1$$

guarantees that S defines a satisfying assignment of ϕ.

(e) For $i \in \{1, \ldots, 2n\}$ the value v_i is a string of $\{0,1\}^{M+N}$ which is interpreted as a decimal number. The first N digits encode the variable corresponding to the literal associated with i: there is exactly one 1 at position i. The last M digits of v_i encode the clauses which contain the literal associated with i: we write an 1 at position $N + k \in \{1, 2, \ldots, M\}$ if and only if the k-th clause contains the literal.

W.r.t. to $\phi = (x_1 \vee \neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3)$ we have:

$$\begin{array}{lll} v_1 = 100\,11 & v_2 = 010\,11 & v_3 = 001\,01 \\ v_4 = 100\,10 & v_5 = 010\,00 & v_6 = 001\,00 \end{array}$$

Consider the satisfying assignment $x_1 = 1, x_2 = 0, x_3 = 1$. The obvious way to produce from it a selection S is to set $S = \{1, 5, 3\}$ – S simply contains those literals which evaluate to true under the assignment. We then have

$$\sum_{i \in S} v_i = 100\,11 + 010\,00 + 001\,01 = 111\,12 \leq 111\,33 = V = W.$$

Obviously, S is not yet a solution of the KNAPSACK instance. In particular, we cannot use any item $i \in \{1, 2, \ldots, 2n\}$ to extend S to a solution as every such v_i also increases one of the last n digits of the sum by one.

Here, the additional items $2n+1, \ldots, 2n+2m$ come into play: for every clause $k = \{1, \ldots, m\}$ we define the values v_{2n+k} and v_{2n+m+k}: the $N+k$-th digit of v_{2N+k} is 1, all other digits are 0; similarly, the only nonzero digit of v_{2N+M+k} is digit $N + k$ which is 2.

In our example this leads to:

$$\begin{array}{ll} v_7 = 000\,10 & v_8 = 000\,01 \\ v_9 = 000\,20 & v_{10} = 000\,02 \end{array}$$

Using these additional items, we can extend our selection S to a solution S' of the KNAPSACK instance. In fact, as we can select a given item at most once, this extension is unique $S' = S \cup \{9, 8\}$.

$$\sum_{i \in S'} v_i = 111\,12 + 000\,20 + 000\,01 = 111\,33 = V.$$

Exercise 3.7

We define SUDOKU to be the following problem: You are given a $n^2 \times n^2$ grid where every entry is either blank or contains a numbers from $\{1, 2, \ldots, n^2\}$. The goal is to decided whether the remaining blank entries of the grid can be labeled by numbers from $\{1, 2, \ldots, n^2\}$ in such a way that every number of $\{1, 2, \ldots, n^2\}$ appears exactly once in (i) every row, (ii) every column, and (iii) in each of the n^2 subgrids.

- Give a reduction SUDOKU $\leq_p$ SAT.

  In particular, apply your reduction to the following SUDOKU instance:

1		2	
			4
	3		
		1	

Remark: One can show that SUDOKU is also **NP**-complete. The adventurous might like to attempt this!