Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Philipp Bamberger, Yannic Maus

Theoretical Computer Science - Bridging Course
Summer Term 2018
Exercise Sheet 8

for getting feedback submit (electronically) before the start of the tutorial on 17th of December 2018.

Exercise 1: The class P

Show that the following languages are in P.

(a) 5-Cycle $= \{\langle G \rangle \mid G$ is a graph and contains a cycle of length $5\}$.

Remark: A cycle of length 5 *in* G *are five distinct nodes* $v_0, \dots, v_4$ *such that the edges* $\{v_i, v_{i+1 \mod 5}\}$, $i = 0, \dots, 4$ *exist in* G.

(b) $L = \{a^n b^{3n} \mid n \geq 0\}$

(c) 17-Independent Set $= \{\langle G \rangle \mid G$ is a graph and contains an independent set of size $17\}$.

Remark: An independent set of a graph with size s *is a set* $S \subseteq V$, $|S| = s$ *such that* $\{v, w\} \notin E$ *for all* $u, w \in S$.

(d) Find a proper citation (e.g., via google) which states whether Primes$=\{\langle n \rangle \mid n \in \mathbb{N}$ is prime$\}$ is in P or not.

Sample Solution

(a) We show the result by giving an algorithm with polynomial runtime. We assume that the graph is stored with an adjacency matrix (one can switch between adjacency matrices and adjacency lists in polynomial time). Then we simply iterate through all 5-tuples $v_1, \dots, v_5$ of nodes in V and for each of them we test whether all the 5 edges $\{v_i, v_{i+1 \mod 5}\}$, $i = 0, \dots, 4$ are there. If this is the case for any of the 5-tuples we return that the graph contains a cycle otherwise we say that it does not. The test for a single edge takes time $O(1)$. Hence the test for a single five tuple takes time $5 \cdot O(1) = O(1)$. As there are at most $\binom{|V|}{5} \leq |V|^5$ different 5 tuples the total runtime is upper bounded by $O(|V|^5)$, which is polynomial in the input length.

(b) It is not difficult to construct a pushdown automaton for the language which works as follows: First it pushes three As to the stack when reading an a and then it removes a single A from the stack with every b (it does not accept any a after it has read the first b). The automaton accepts the input if the stack is empty at the end of the input. This pushdown automaton can immediately be simulated by a 2-band Turing machine (where the second band works as the stack), which shows that the language is in $\mathcal{P}$

(c) We show the result by giving an algorithm with polynomial runtime. We assume that the graph is stored with an adjacency matrix. Then we simply iterate through all 17-tuples $v_1, \ldots, v_{17}$ of nodes in V and for each of them we test whether none of the edges $\{v_i, v_j\}, i \neq j$ exist. If this is the case for any of the 17 tuples we return true, otherwise false.

For a single 17-tuple there are $\binom{17}{2} \leq 17^2$ edges to be tested. As we can test for a single edge in time $O(1)$ the test for a single tuple takes time $17^2 \cdot O(1) = O(1)$. There are at most $\binom{|V|}{17} \leq |V|^{17}$ tuples. Thus the runtime is upper bounded by $|V|^{17} \cdot O(1) = O(|V|^{17})$, which is polynomial.

(d) E.g., the journal version of the original paper. *Agrawal, Manindra; Kayal, Neeraj; Saxena, Nitin (2004). PRIMES is in P. Annals of Mathematics. 160 (2): 781-793.*

Repetition of Course Material *(0 Points)*

Let L_1, L_2 be languages (problems) over alphabets Σ_1, Σ_2. Then $L_1 \leq_p L_2$ (L_1 is polynomially reducible to L_2), iff a function $f : \Sigma_1^* \to \Sigma_2^*$ exists, that can be calculated in polynomial time and

$$\forall s \in \Sigma_1 : s \in L_1 \iff f(s) \in L_2.$$

Language L is called $\mathcal{NP}$-hard, if *all* languages $L' \in \mathcal{NP}$ are polynomially reducible to L, i.e.

$$L \ \mathcal{NP}\text{-hard} \iff \forall L' \in \mathcal{NP} : L' \leq_p L.$$

The reduction relation '$\leq_p$' is transitive ($L_1 \leq_p L_2$ and $L_2 \leq_p L_3 \Rightarrow L_1 \leq_p L_3$). Therefore, in order to show that L is $\mathcal{NP}$-hard, it suffices to reduce a known $\mathcal{NP}$-hard problem $\tilde{L}$ to L, i.e. $\tilde{L} \leq_p L$. Finally a language is called $\mathcal{NP}$-complete ($\Leftrightarrow$: $L \in \mathcal{NPC}$), if

1. $L \in \mathcal{NP}$ and
2. L is $\mathcal{NP}$-hard.

Exercise 2: The Class $\mathcal{NPC}$ *(7 Points)*

This exercise (and similar ones) is really (!!) important for the course.

Show HittingSet$:=\{\langle \mathcal{U}, S, k\rangle \,|\, \text{universe } \mathcal{U} \text{ has subset of size} \leq k \text{ that } \textit{hits} \text{ all sets in } S \subseteq 2^{\mathcal{U}}\} \in \mathcal{NPC}$.[1]

Use that VertexCover $:= \{\langle G, k\rangle \mid \text{Graph } G \text{ has a } \textit{vertex cover} \text{ of size at most } k\} \in \mathcal{NPC}$.

*Remark: A **hitting set** $H \subseteq \mathcal{U}$ for a given universe $\mathcal{U}$ and a set $S = \{S_1, S_2, \ldots, S_m\}$ of subsets $S_i \subseteq \mathcal{U}$, fulfills the property $H \cap S_i \neq \emptyset$ for $1 \leq i \leq m$ (H 'hits' at least one element of every S_i).*
*A **vertex cover** is a subset $V' \subseteq V$ of nodes of $G = (V, E)$ such that every edge of G is adjacent to a node in the subset.*

Hint: For the poly. transformation ($\leq_p$) you have to describe an algorithm (with poly. run-time!) that transforms an instance $\langle G, k\rangle$ of VertexCover *into an instance $\langle \mathcal{U}, S, k\rangle$ of* HittingSet*, s.t. a vertex cover of size $\leq k$ in G becomes a hitting set of $\mathcal{U}$ of size $\leq k$ for S and vice versa(!).*

Sample Solution

We first show that hitting set belongs in $\mathcal{NP}$, by engineering a deterministic polynomial time verifier for it. Then we will prove that it is an $\mathcal{NP}$-hard problem, by reducing a known $\mathcal{NP}$-hard problem, vertex cover (as mentioned in the hint), to hitting set in polynomial time.

Guess and Check: Given a finite set $\mathcal{U}$, a collection S of subsets of $\mathcal{U}$, a positive integer k and a finite set H as a certificate, the following deterministic polynomial time verifier for hitting set confirms

[1] The power set $2^{\mathcal{U}}$ of some ground set $\mathcal{U}$ is the set of *all subsets* of $\mathcal{U}$. So $S \subseteq 2^{\mathcal{U}}$ is a collection of subsets of $\mathcal{U}$.

in polynomial time that $(\mathcal{U}, S)$ has a hitting set of size at most k. Let λ be the sum of the sizes of all the subsets S_i in S and δ the size of $\mathcal{U}$. Note that we can check if A is a subset of B with the following brute-force algorithm: $\forall a \in A$ check if $\exists b \in B : a = b$ which needs $\mathcal{O}(|A| \cdot |B|)$ comparisons. We can check if H is a subset of $\mathcal{U}$ that has at most k elements with $\mathcal{O}(k \cdot \delta)$ comparisons and if it contains at least one element from each subset S_i in the collection S, with $\mathcal{O}(\lambda \cdot k)$ comparisons. We accept iff both checks are true. These two checks are obviously equivalent to the problem's definition, so hitting set has a polynomial time verifier. Therefore it belongs in $\mathcal{NP}$.

Polynomial Reduction of VertexCover **to** HittingSet**:** We will create a polynomial time reduction from vertex cover to hitting set, proving that since vertex cover is $\mathcal{NP}$-hard, hitting set must also be $\mathcal{NP}$-hard.
The reduction takes as input an undirected graph $G = (V, E)$, where V is a set of nodes and E a set of edges defined over those nodes, as well as a positive integer k and outputs the set V, the collection $E = \{e_1, e_2, \ldots, e_n\}$ of subsets of V and the positive integer k. We claim the following equivalence holds:

"G has a vertex cover of size at most k" $\Leftrightarrow$ "(V, E) has a hitting set of size at most k"

Here is the proof:

"G has a vertex cover of size at most k" $\Leftrightarrow$
$\exists V' \subseteq V : |V'| \leq k$ and $\forall$ edge $e_i = \{u_i, v_i\} \in E, u_i \in V'$ or $v_i \in V' \Leftrightarrow$
$\exists V' \subseteq V : |V'| \leq k$ and $\forall$ subset e_i in collection E $\exists c \in e_i : c \in V' \Leftrightarrow$
"(V, E) has a hitting set of size at most k"

This reduction takes time linear to the size of the input (all it does is copy the input to the output), therefore polynomial. Also, as we showed, it is correct. Therefore, hitting set is at least as hard as vertex cover and since vertex cover is $\mathcal{NP}$-hard, so is hitting set.
One might notice that this reduction was rather straightforward. This makes sense, since vertex cover is a special version of hitting set, where each subset S_i in the collection S has exactly two elements of $\mathcal{U}$. Obviously, no problem can be harder than its generalization and since vertex cover is $\mathcal{NP}$-hard, hitting set (as a generalization of vertex cover) must also be $\mathcal{NP}$-hard.