Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
P. Bamberger, Y. Maus,

Theoretical Computer Science - Bridging Course
Summer Term 2017
Exercise Sheet 8

Hand in (electronically or hard copy) by 12:15 pm, December 18th, 2017

Exercise 1:

Let Σ be a fixed finite alphabet. Show that the language of deterministic finite automatons (DFAs) on Σ that accept every word is decidable. Formally, show that

$$L = \{\langle A\rangle \mid A \text{ is a deterministic finite automaton with } L(A) = \Sigma^*\}$$

is a decidable language.
Remark: You can use that it is not difficult to construct a Turing machine which tests whether an input is the well formed encoding of a deterministic finite automaton.

Sample Solution

Let B be a DFA such that the language generated by B is Σ^*. That is, $L_B = \Sigma^*$. (It is easy to see that this is always possible for any given alphabet Σ.) We have shown in the video lecture that testing equivalence for two DFA is a decidable problem. Let M be such a turing machine that can test equivalence for two DFA. We can construct a turing machine M', such that upon input A where A is a DFA, it will run M with input $\langle A, B\rangle$. If M accepts the input, then M' accepts the input A as well, otherwise M' rejects. Since M will give definite answer in finite time, we know M' will give definite answer in finite time as well. Hence, we know $\mathcal{L}$ is decidable.

Exercise 2: 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 $v_0, \ldots, v_4$ such that the edges $\{v_i, v_{i+1 \bmod 5}\}$, $i = 0, \ldots, 4$ 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 $v_1, \ldots, v_5$ of nodes in V and for each of them we test whether all the 5 edges $\{v_i, v_{i+1 \bmod 5}\}$, $i = 0, \ldots, 4$ 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.*

Exercise 3: Decision vs. construction?

In this exercise we want to show the relation of the *decision* and the *optimization* variant of the knapsack problem.
The *knapsack problem (KP)* is defined as follows. Input: $b \in \mathbb{N}$, weights $w_1, \ldots, w_N \in \{1, \ldots, b\}$ and profits $p_1, \ldots, p_N \in \mathbb{N}$. A feasible solution is a $K \subseteq \{1, \ldots, N\}$ such that

$$\sum_{i \in K} w_i \leq b \ .$$

In the *optimization variant* the goal is to find a feasible solution which maximizes the profit

$$\sum_{i \in K} p_i \ .$$

We can define a *decision variant* of the problem as follows:

$$KP = \{\langle b, w_1, \ldots, w_N, p_1, \ldots, p_N, k \rangle \mid b, w_1, \ldots, w_N, p_1, \ldots, p_N \text{ forms a valid knapsack instance and it has a feasible solution with profit at least } k\}.$$

It is well known that the decision variant of KP is an $\mathcal{NP}$-complete problem.
In the following we will (implicitly) show that the optimization variant is also $\mathcal{NP}$-complete.

(a) Show how to use a polynomial number of repetitions of a KP algorithm to determine the optimal value (profit) of a feasible solution for a given knapsack instance.

(b) Use the previous result to show that if the decision variant of KP is in $\mathcal{P}$ then the optimization variant (=computing an optimal solution) of KP is in $\mathcal{P}$ as well.

Hint: Try to use the algorithm from a) to iteratively decide whether an item i is needed in an optimal solution.

Sample Solution

(a) We use a binary search (cf. `https://en.wikipedia.org/wiki/Binary_search_algorithm`) to find the optimal value in the interval $[0, P]$ with $P = \sum_{i=1}^{N} p_i$. Note that the maximal profit is upper bounded by P.

We begin with $p = P/2$ and use the decision variant of KP to see where we have to continue with the search. This uses at most $\lceil \log_2 P + 1 \rceil$ (=polynomially many) iterations of the decision variant.

Remark: Note that a linear search takes time linear in the size of the search space, i.e., it would take $O(P)$ time in the above setting. As usually the p_i are given by a binary encoding (e.g. the value 2051 is given by 100000000011). The binary encoding of a number n only uses $\lceil \log_2 n + 1 \rceil$ bits, or in other words the absolute value of a number in binary encoding is exponential in the length of the encoding (in the example above the number is 2051 and the length of the encoding is only 12). Thus the absolute value of P can be exponential in the input length and a linear search would simply take too long.

(b) We denote the algorithm (from (a)) which computes the optimal value for the knapsack instance with item set K as $B(K)$. Then we can solve the construction version of the problem in polynomial time with the following algorithm.

```
1: $K := \{1, \ldots, N\}$
2: $p = B(K)$
3: **for** i=1 to N **do**
4:     **if** $B(K \setminus \{i\}) = p$ **then** $K := K - \{i\}$
5: **Output** $K$
```

That is, we begin with all a set K of all items and iterate through the items to determine which items are needed for an optimal solution. For each item we test whether the optimal value stays the same if we were to remove the item (note that the optimal value cannot increase by removing an item). If we can still achieve the same optimal value without using the item we do not need it and remove it from the list. In the end we return the items that remain.