Tutorial 10

Exercise 1 (compulsory)

Below are some definitions of the class NP. Which ones are correct?

1. NP is the class of languages which have polynomial time verifiers.
2. NP is the class of languages that cannot be decided in polynomial time using a deterministic Turing machine.
3. NP is the class of languages that have nondeterministic verifiers.
4. NP is the class of languages that can be decided in polynomial time on a nondeterministic Turing machine.

Solution:

1. Correct.
2. Incorrect.
3. Incorrect.
4. Correct.

Exercise 2 (compulsory)

Consider the following decision problem, also know as *bin packing*. Given a finite set S of natural numbers, a number k of available bins, and a maximum capacity M of the bins, determine whether the items can be partitioned into $S_1, \ldots, S_k$ such that $\bigcup_{1 \le i \le k} S_i = S$ and the sum of the items in each bin does not exceed the maximum capacity M, i.e., $\sum S_i \le M$ for all i, $1 \le i \le k$.

1. Define a language $BINPACK$ for the above mentioned decision problem.
2. Argue that $BINPACK \in$ NP by constructing a polynomial time nondeterministic TM deciding it.
3. Argue that $BINPACK \in$ NP by constructing a polynomial time verifier for $BINPACK$.

Solution:

1. $$BINPACK \overset{\text{def}}{=} \{\langle S, k, M\rangle \mid S \subseteq \mathbb{N} \text{ is a finite set, } k, M \in \mathbb{N}\text{, and there is a partition } S_1 \cup S_2 \cup \ldots \cup S_k = S \text{ such that } \textstyle\sum S_i \le M \text{ for all } i,\ 1 \le i \le k\ \}$$
2. "On input $\langle S, k, M\rangle$:
   1. Nondeterministically split S into $S_1, \ldots, S_k$.
   2. For all i check whether $\sum S_i \le M$.
   3. If the check was successful then <u>accept</u>, else <u>reject</u>."

   Clearly both steps 1. and 2. take only polynomial time, step 3. takes only constant time, so the algorithm runs in *nondeterministic* polynomial time.
3. "On input $\langle\langle S, k, M\rangle, c\rangle$:
   1. Verify whether c encodes a partitioning $S_1, \ldots, S_k$ of S. If not then <u>reject</u>.
   2. For all i check whether $\sum S_i \le M$.
   3. If the check was successful then <u>accept</u>, else <u>reject</u>."

   Clearly both steps 1. and 2. take only polynomial time, step 3. takes only constant time, so the verifier runs in *deterministic* polynomial time.

Exercise 3 (compulsory)

Let $PATH \stackrel{\text{def}}{=} \{\langle G, s, t\rangle \mid G$ is a graph, s and t two nodes in the graph and there is a path from s to t $\}$.

Each of the following two algorithms claim to decide $PATH$ in nondeterministic polynomial time (and hence conclude that $PATH$ belongs to NP). However, they both contain an error. Find it and explain what is wrong.

1. "On input $\langle G, s, t\rangle$:
   1. Nondeterministically select a number $k \in \mathbb{N}$. (The length of a path from s to t.)
   2. Nondeterministically select k nodes in the graph G.
   3. Verify whether the first node is s, the last one is t, and they are all connected by edges.
   4. If yes, then accept, else reject."

2. "On input $\langle G, s, t\rangle$ of length n:
   1. Nondeterministically select a number $k \le 2^n$.
      (Note that k written in binary is of polynomial length w.r.t. to n.)
   2. Nondeterministically select k nodes in the graph G.
   3. Verify whether the first node is s, the last one is t, and they are all connected by edges.
   4. If yes, then accept, else reject."

Because you found an error in both of the algorithms, can you so conclude that $PATH \notin$ NP?

Solution:

1. This Turing machine is not even a decider! The problem is in step 1. We agreed that numbers will be written in binary on a tape, but there is no bound on the length of the number k. Hence we can nondeterministically keep generating the symbols 0 and 1 on the tape and at any time nondeterministically decide to continue with step 2. However, we cannot force the machine to even go to step 2., which means that there will be always one infinite branch in the computation tree that will keep generating the bits of the number k for ever without ever entering the step 2.

2. This time the Turing machine is a decider for $PATH$ but its running time is not polynomial. In fact, the step number 1. is now OK (it takes only polynomial time because k is written in binary). Nevertheless, the problem is now in step number 2. which will in the worst case write 2^n nodes on a tape (some of them will of course repeat) and this takes exponential time.

Based on the fact that the two algorithms above are wrong, we cannot of course conclude that $PATH \notin$ NP. In fact $PATH$ belongs even to the class P because it can be solved in deterministic polynomial time using e.g. depth-first search.

Exercise 4 (compulsory)

Prove that co-NP $\subseteq$ EXPTIME.

Solution:
Let $L \in$ co-NP. We want to show that $L \in$ EXPTIME. Because $L \in$ co-NP then by definition $\overline{L} \in$ NP. This means that there is a nondeterministic decider M for $\overline{L}$ running in time $O(n^k)$ for some constant k. By the determinization theorem there is an equivalent deterministic decider for $\overline{L}$ running in time $2^{O(n^k)}$. Because deterministic deciders can be complemented by swapping the accept and reject state, we have also a deterministic decider for L running in the same time $2^{O(n^k)}$. This by definition means that $L \in$ EXPTIME.

Exercise 5 (compulsory)

Prove that $\leq_P$ is transitive. In other words show that if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$ then $L_1 \leq_P L_3$. Do not forget to carry on a complexity analysis of the construction.

Solution:
Assume that $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$. We want to show that $L_1 \leq_P L_3$, i.e., find a polynomial-time computable function f such that $w \in L_1 \iff f(w) \in L_3$. We know that $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, so there exist polynomial time computable functions g and h such that $w \in L_1 \iff g(w) \in L_2$ and $w \in L_2 \iff h(w) \in L_3$. So define f by $f(u) = h(g(u))$.

We must now show that f is polynomial time computable. We know that g is polynomial time computable by a Turing machine M_g with running time $O(n^k)$ and that h is computable using a Turing machine M_h with running time $O(n^\ell)$. We can now compute f using the following Turing machine M_f:

"On input w:

1. Run M_g on input w.
2. Run M_h on input $g(w)$.
3. As the final output, output the string returned by M_h."

Let $n = |w|$. Stage 1. requires $O(n^k)$ steps. Since every step of stage 1. may produce a new character, we have that $|g(w)| = O(n^k)$, and therefore stage 2. may require up to $O((n^k)^\ell) = O(n^{k\ell})$ steps. All in all we therefore use $O(n^k + n^{k\ell}) = O(n^{k\ell})$ steps, i.e., M_f has a polynomial running time.

Exercise 6 (optional, but interesting)

Consider the decision problem

Given an undirected graph G, is it the case that G has a clique of size 4?

1. Express this problem as the language *FOURCLIQUE*.
2. Prove that *FOURCLIQUE* $\in$ NP.
3. Prove that *FOURCLIQUE* $\in$ P.
4. Do we now know that *CLIQUE* $\in$ P? Justify your answer.

Solution:

1. The language is

$$\textit{FOURCLIQUE} = \{\langle G\rangle \mid G \text{ is an undirected graph with a clique of size } 4\}$$

2. We can show that *FOURCLIQUE* $\in$ NP in two different ways:

By constructing a polynomial time verifier: The certificate is here a clique of size 4. The verifier is

On input $\langle G, c\rangle$:
(a) If c is not a set of 4 nodes from G then <u>reject</u>
(b) Else check that every pair of nodes is connected by an edge.
(c) If this is the case, then <u>accept</u>, else <u>reject</u>.

Let n denote the size of G, where $G = (V, E)$, i.e. $n = |V| + |E|$. Stage 1. may require up to $O(n)$ steps. Stage 2. requires up to $O(n)$ steps. The verifier uses a total of $O(n) + O(n) = O(n)$ steps and therefore has polynomial time complexity.

By constructing a polynomial time NTM: A nondeterministic decider for *FOURCLIQUE* is

```
On input $\langle G \rangle$:
(a) Guess $C$, a set of 4 nodes from $G$.
(b) Check that each pair of nodes from $C$ is connected by an edge.
(c) If this is the case, then accept else reject
```

The running time of this nondeterministic decider is polynomial, which we see as follows: Again, let n denote the size of G where $G = (V, E)$, i.e. $n = |V| + |E|$. Stage 1. requires $O(n)$ steps. Stage 2. requires at most $O(n)$ steps. The verifier therefore uses $O(n)$ steps.

3. In a graph with v nodes there are

$$\left(\begin{array}{c} v \\ 4 \end{array} \right) = \frac{v!}{(v-4)!4!} = v(v-1)(v-2)(v-3) = O(v^4)$$

sets of nodes with exactly 4 elements. We can therefore create the following polynomial-time algorithm for *FOURCLIQUE*:

```
On input $\langle G \rangle$:
  1. For every set $C$ of $4$ nodes from $G$:
       • Check that every pair of nodes from $C$ is connected by an edge.
       • If this is the case, then accept.
  2. If no set of $4$ nodes was a clique, then reject.
```

Given a graph $G = (V, E)$ let $v = |V|$ and $e = |E|$ and $n = v + e$. The algorithm will perform at most $O(v^4)$ traversals of its main loop. Every traversal will require at most $O(e)$ steps, since each edge must be examined at most once. The algorithm therefore uses at most $O(ev^4) = O(n^5)$ steps and therefore has polynomial time complexity.

4. We get no information about *CLIQUE* from our knowledge that *FOURCLIQUE* $\in$ P, since *CLIQUE* is a different problem. It is defined by

$$\textit{CLIQUE} = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a clique of size } k\}$$

The problem therefore has two parameters, while *FOURCLIQUE* only has one.