Computability and Computational Complexity, A.Y. 2024–2025

Guide to the answers

Monday, July 21, 2025

Exercise 1

1.1) Given a finite alphabet and using deterministic Turing machines as the underlying computational model, write the definitions of *Recursive* (R) and *Recursively Enumerable* (RE) language.

1.2) Prove that the definition of recursive language does not change if we use non-deterministic TMs (i.e., every language that is recursive wrt DTMs is also recursive wrt NDTMs and vice versa).

Solution 1

1.1) See the notes.

1.2) We know that, given enough time, a DTM can simulate a NDTM. Therefore, if there is a NDTM that accepts a language, a DTM can simulate it by executing sequentially all possible branches of the computation, accepting if a computation halts in acceptance and rejecting if none accepts. The opposite is also true, since a DTM can be seen as a special case of NDTM with one branch per step.

Exercise 2

In order to schedule a (very short) exam session, a University department must distribute n exams among $m = 3$ time slots. In general n can be very large, therefore many exams must take place at the same time.
To try to avoid conflicts, all students are asked to register to exams beforehand; two exams are *conflicting* if at least one student is registered to both. The department will try to assign conflicting exams to different time slots, so that no student has to sit through two exams at the same time. If such a schedule exists, we call it *non-conflicting*.

2.1) Prove that the problem of deciding the existence of a non-conflicting schedule is in **NP**. In particular, clarify what is the input and what is its size with respect to n (remember that m is fixed to 3).

2.2) Prove that, as the number n of exams grows, the university policy is not scalable, in the sense that there is no known algorithm that can determine the existence of a non-conflicting exam schedule in polynomial time with respect to n (again, remember that $m = 3$).

2.3) Provide polynomial algorithms to decide the problem when $m = 1$, $m = n-1$ and $m \geq n$. Is the problem still polynomial wrt the number n of exams when $m = 2$?

Hint — *As usual, here is a list of known **NP**-complete problems for reference: SATISFIABILITY, 3-SATISFIABILITY, CLIQUE, INDEPENDENT SET, INTEGER LINEAR PROGRAMMING, VERTEX COVER, 3-VERTEX COLORING, SUBSET SUM, KNAPSACK, HAMILTONIAN PATH, DIRECTED HAMILTONIAN CYCLE, HAMILTONIAN CYCLE, TRAVELING SALESMAN PROBLEM.*

Solution 2

2.1) We can represent the exams as nodes in a graph, with links between each pair of conflicting exams. The input to our algorithm is therefore, as in any graph problem, an adjacency list or an adjacency matrix whose size is a low-degree polynomial wrt n.
The certificate is a list of assignments of exams to slots (basically, a list of n numbers from $\{1, 2, 3\}$) whose size is again polynomial wrt n. To verify the assignments, we check that no conflicting exams receive the same slot; again this is solved with scans of the input and of the certificate.

2.2) The problem is a rephrasing of 3-VERTEX COLORING, where the graph of conflicts must be colored with $m = 3$ colors so that no two conflicting (i.e., connected) nodes have the same slot (color).
More formally, we can reduce an instance of 3-VERTEX COLORING to our problem by the correspondence written above; given that 3-VERTEX COLORING is **NP**-complete, our problem is too (since we already proved that it is **NP**).

2.3) If we only have $m = 1$ time slot, the only way to have a non-conflicting schedule is when no two exams are in conflict, which can be checked with a scan of the input;
if we have $m = n - 1$ time slots, we just need to find two exams that have no common student, assign them to the same time slot, then assign all other exams to a new time slot; if all exams afre in conflict, then the schedule is impossible;
if the number of slots is equal or larger than the number of exams, we can assign every exam to its own slot to avoid conflicts.
Finally, if $m = 2$ we have an instance of 2-VERTEX COLORING, which is known to be polynomial wrt input size.
In all the discussed cases, the problem is tractable.

Exercise 3

Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ be a three-symbol alphabet. Consider the language $L \subset \Sigma^*$ of strings where the three symbols have the same number of occurrences. For example:

$$\mathtt{abbcaccba} \in L \qquad \varepsilon \in L$$
$$\mathtt{abc} \in L \qquad \mathtt{aabb} \notin L$$
$$\mathtt{abacab} \notin L \qquad \mathtt{aaabbbccc} \in L$$

3.1) Prove that $L \in \mathbf{P}$.

3.2) Prove that $L \in \mathbf{L}$.

Solution 3

3.1) Initialize 3 counters to 0; scan the input and increment the first counter upon finding an $\mathtt{a}$, the second counter upon finding a $\mathtt{b}$, or the third counter upon finding a $\mathtt{c}$.
At the end of the scan compare the three counters: if they are equal accept, otherwise reject. Since the algorithm requires a simple scan of the input, with a (logarithmic-time) increment of a counter at each step, and finally two (logarithmic-time) comparisons, the whole setup is polynomial-time wrt the input size.

3.2) The algorithm requires maintaining three logarithmic-sized counters (wrt input size), therefore it belongs to **L**.