Computability and Computational Complexity, A.Y. 2023–2024

Guide to the answers

Thursday, August 29, 2024

Exercise 1

1.1) Define the space complexity class **L**.

1.2) Consider the following language of strings in $\{\mathtt{a},\mathtt{b}\}^*$:

$$L = \{\mathtt{a}^{n_1}\mathtt{b}\mathtt{a}^{n_2}\mathtt{b}\mathtt{a}^{n_3}\mathtt{b}\cdots\mathtt{b}\mathtt{a}^{n_k} \;:\; k \in \mathbb{N} \wedge k \geq 1 \wedge n_1 > n_2 > n_3 > \cdots > n_k > 0\}.$$

Namely, a string is in L iff it is composed of non-empty sequences of a's of decreasing length separated by single b's, e.g.:

$\mathtt{aaaaabaaaba} \in L$, $\mathtt{aaba} \in L$,
$\mathtt{aaabaaa} \notin L$ (two subsequences with the same number of a's: $n_1 = n_2 = 3$),
$\mathtt{aaabaaba} \in L$, $\mathtt{aaaaaaaaaaaaaaba} \in L$,
$\mathtt{aaabbaa} \notin L$ (two consecutive b's),
$\mathtt{aabaaa} \notin L$ (the sequences of b's are not of decreasing length).

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

Hint — *For 1.2: start by sketching down a (pseudocode) program that decides L.*

Solution 1

1.1) See the lecture notes. Some clarity in defining what "space" is in the definition's context is appreciated.

1.2) An algorithm to check if a string $s \in \{\mathtt{a},\mathtt{b}\}^*$ belongs to L just needs to scan the string, remember the length of the latest full sequence of a's it has found, counting the length of the current one, and rejecting as soon as one of the definition's constraints is violated:

```
1.  on input s ∈ {a,b}*
2.      current_size ← 0                          Size of the current sequence of a's
3.      last_size ← 0                             Size of the previous sequence of a's
4.      for c in s                                scan all symbols in the string
5.          if c = 'a'                            We are in a sequence of a's
6.              current_size ← current_size + 1   increase the counter
7.              if last_size > 0                  if this is not the first such sequence...
8.                      and current_size ≥ last_size   ...and it's longer than the last one...
9.                  reject                        ...then s ∉ L
10.         else                                  Found b: a sequence of a's has been completed
11.             if current_size = 0               if it was empty...
12.                 reject                        ...then s ∉ L
13.                                               Otherwise we can continue
14.             last_size ← current_size          Remember the size of the last sequence of a's
15.             current_size ← 0                  reset the counter for the next sequence of a's
16.     if current_size = 0                       if the last sequence of a's was empty
17.         reject                                ...then s ∉ L
18.     accept                                    if no violation was found, then s ∈ L.
```

The above code uses two counters, *current_length* (playing the role of n_i for the ith sequence of `a`'s being scanned) and *last_length*, storing n_{i-1}. Since the algorithm only uses two counters, each needing to store a number no larger than the size of s, it is clear that the space required is $O(\log |s|)$.

Observations

The code above is just an example; I haven't checked its correctness and some edge cases might still be missing. Errors such as accepting zero-length sequences of `a`'s, or forgetting the equality in a comparison were not taken into account; purely verbal answers would be accepted, provided that they were free of ambiguities.
Moreover, observe that a TM implementation would require more details, e.g., at least a three-symbol alphabet to delimit the input string with blanks and to separate counters in the working tape, but it should be apparent that such details would not compromise the main argument that two logarithmic counters are sufficient.

Exercise 2

A teacher wants to divide a large class of N students into a small number $k \leq N$ of groups, each to be assigned a different project. She sets a constraint on how groups are formed:

No-past-collaboration constraint*: Every group must be composed of students who never collaborated before — i.e., if two students already collaborated in a previous group project, then they must be placed in different groups.*

The teacher wants to know if she has prepared enough projects to be able to satisfy the constraint. Luckily for her, the College's Statistical Service maintains a comprehensive list of student groups formed in the past. Thus, she can formally define the following decision problem.

*Given (1) the number N of students, (2) the list of past student collaborations (e.g., in the form of a list pairs of students) and (3) the number k of available projects, is it possible to split the N students into k groups so that the **no-past-collaboration** constraint is satisfied?*

2.1) What complexity class does the above defined decision problem belong to, and why?

2.2) In particular, for what values of k does an efficient decision procedure exist? Among them, for what values of k is the decision trivial?
Note that group sizes need not be balanced, nor are we required to actually create the groups: we only need to decide if the number k of projects is enough, or if the teacher needs to devise more of them.

Solution 2

2.1) Our problem is clearly in **NP**, since a solution (a partition of the students into k groups) can be verified in polynomial time.
We can also prove its **NP**-completeness: once the set of students and past collaborations has been encoded into a graph, the problem becomes that of assigning to each node a label from the set $\{1, 2, \ldots, k\}$ so that no connected nodes have the same label. Therefore, for a fixed value of k, the problem is equivalent to k-VERTEX COLORING. In other words, we can (polynomially) reduce any instance of k-VERTEX COLORING to our problem. Since k-VERTEX COLORING is **NP**-complete for $k \geq 3$, then our problem, for which k is an arbitrary input value, is **NP**-complete.

2.2) However, for *specific* combinations of N and k the problem may have a polynomial (or even trivial) solution:

- $k = 0$: only possible if $N = 0$ (trivial);
- $k = 1$: only possible if the graph is completely disconnected (polynomial);
- $k = 2$: equivalent to 2-VERTEX COLORING (polynomial);
- $k = N - 1$: always possible unless the past collaboration graph is complete (polynomial):
- $k \geq N$: always possible (trivial).

By "trivial" I mean that we don't need to look at the adjacency structure at all. "Trivial" cases still require a check on the values of k and N, and should therefore be regarded as linear (i.e., still polynomial) or at best logarithmic (since the representations of k and N are logarithic wrt the input size dominated by the adjacency list/matrix).

Observations

After looking at student answers, I realized that the problem description could mislead into trying to prove some equivalence to the INDEPENDENT SET problem. Indeed, every single group is an independent set within the graph; however, here we are not requiring an independent set of size k, but a partition of the graph into k independent sets, which is precisely k-VERTEX COLORING.

Students who fell in this (unintentional) trapdoor have not been penalized for that, and the answer was scored on the soundness of their analysis, even if a final correct answer could not be obtained. For instance, an iterated greedy search for the largest independent set is an interesting proposal.

Exercise 3

3.1) State Rice's Theorem about the (un-) decidability of Turing Machine properties as precisely as possible.

3.2) Outline a proof of the Theorem.

Solution 3

3.1) See the lecture notes. A correct definition of "non-trivial" and "semantic" was clearly necessary before stating the theorem. however, statements that omitted those definitions were accepted if a correct outline of the proof in 3.2 was detailed enough to prove the knowledge of those definitions.

3.2) See the lecture notes.