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

Guide to the answers

Friday, July 19, 2024

Exercise 1

1.1) Define the **PSPACE** complexity class.

1.2) Prove **P** $\subseteq$ **PSPACE**.

1.3) Prove **NP** $\subseteq$ **PSPACE**.

Solution 1

See the lecture notes. In particular, point 1.3 can be proved either by considering the additional space required by the simulation of a NDTM by a DTM, or by invoking Savitch's Theorem (whose immediate consequence is that **PSPACE** $=$ **NPSPACE**).

Exercise 2

The FACTORING decision problem is the following: given a pair of numbers $(n, k) \in \mathbb{N}^2$, does n have a prime factor larger than or equal to k?
For instance, $(28, 5) \in$ FACTORING because 28 is divisible by 7, which is a prime number larger than 5.
On the other hand, $(27, 5) \notin$ FACTORING because the only prime that divides 27 is 3, which is less than 5.

2.1) Prove that FACTORING $\in$ **PSPACE**.

Hint — *This can be achieved in two ways.*
*You can prove it directly, by providing a simple algorithm that scans all prime numbers larger than k and checks if any of them is a divisor of n, and showing that this algorithm is **PSPACE**.*
*Or you can prove it indirectly by showing that FACTORING belongs to a more convenient complexity class that is a subset of **PSPACE** (e.g., **NP**).*
***Bonus points** if you can give both proofs.*

Solution 2

Direct proof

Given two inputs n and k, and assuming a positional (e.g., base 2 or base 10) representation of integers, the input size is $O(\log n)$ (we can assume that $k < n$, since otherwise the answer is "No").
The following algorithm scans all numbers $p \in \{k, k+1, \ldots, n-1, n\}$ and, for each p, tests if p divides n and, in that case, if p is prime (again, by testing its divisibility by all numbers below it). If all conditions are met, then the answer is "yes"; otherwise, if the scan completes and no such p is found, the answer is "no":

```
function FACTORING (n, k)
    for p ← k ... n
        if n % p = 0
            p_is_prime ← true
            for i ← 2 ... p-1
                if p % i = 0
                    p_is_prime ← false
            if p_is_prime
                accept and halt
    reject and halt
```

The algorithm is very inefficient, since both loops take an exponential time wrt the input size, however it only uses two integer variables (p and i) of size at most n, and a boolean variable. Therefore, the space it uses is linear with respect to the input size.

Indirect proof

It is easy to see that FACTORING $\in$ **NP**, since a number p satisfying the condition is a polynomially verifiable certificate. moreover, we already know (by exercise 1.3) that **NP** $\subseteq$ **PSPACE**.

Exercise 3

3.1) Define Radó's n-state, 2-symbol Busy Beaver.

3.2) On July 2nd, 2024, the 5-state, 2-symbol Busy Beaver has been announced. It halts after $S(5) = 47,176,870$ steps leaving $\Sigma(5) = 4,098$ 1's on the tape.
Being aware of this fact, propose a simple algorithm that decides the empty-tape Halting Problem (the version HALT_ε for machines starting on an empty tape) for all Turing Machines with two symbols and at most 5 states.

Solution 3

3.1) See the notes.

3.2) The Busy Beaver is defined in terms of the number of 1's left on the tape upon halting, however the text clarifies that the same machine is also the longest-running 5-state 2-symbol halting Turing Machine. As such, given any 5-state 2-symbol TM $\mathcal{M}$, we just need to simulate $\mathcal{M}$ for a maximum of 47,176,870 steps. If it halts within this number of steps, then answer "Yes"; if it runs longer, we know that it is going to run forever (otherwise it would beat the Busy Beaver), and we can safely answer "No".