SAPIENZA
UNIVERSITÀ DI ROMA

“Sapienza” University of Rome
Faculty of Information Engineering,
Informatics and Statistics
Department of Computer Science

Computational Complexity

Lecture notes integrated with the book "Computational Complexity: a modern approach", S. Arora, B. Barak

Author
Simone Bianco

September 26, 2024

Contents

Information and Contacts

Personal notes and summaries collected as part of the *Computational Complexity* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
https://github.com/Exyss/university-notes. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **bianco.simone@outlook.it**
- LinkedIn: **Simone Bianco**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

Suggested prerequisites:

Basic knowledge of computability theory and algorithm complexity

Licence:

1 Introduction on Turing Machines

1.1 Turing Machines

Throughout history, humans have been solving problems through a wide variety of models capable of computing valid results, ranging from their intellect to mechanical devices capable of solving problems. In particular, each computational model can be described as a list of sequential operations. Given the same initial conditions, these lists are expected to yield the same exact result each time the computation is executed.

In modern mathematics, this is described through the concept of **algorithm**, which is a finite list of unambiguous instructions that, given some set of initial conditions, can be performed to compute the answer to a given problem. Even though this is a straightforward definition for an algorithm, it isn't as "mathematically stable" as it seems: each computational model could have access to a different set of possible operations, meaning that the same problem could be solved by different kinds of algorithms. In 1950, Alan Turing was able to define a **theoretical computational model** capable of capturing the concept of computation itself through a simple - but sufficient - theoretical machine.

A Turing machine is made of:

- A *tape* divided into cells, where each cell contains a symbol from a finite set called *alphabet*, usually assumed to contain only 0 and 1, or a special symbol $\sqcup$, namely the *blank character*. The tape is finite on the left side but infinite on the right side.
- A *read-write head* capable of reading and writing symbols on the tape. The head is always positioned on a single cell of the tape and can shift left and right only one cell per shift.
- A finite set of *states* that can be assumed by the machine. At all times the machine only knows its current state. The set contains at least one state that is capable of immediately halting the machine when reached (such states could be unreachable, making the machine go in an infinite loop).

- A finite set of *instructions* which, given the current state and the current cell read by the read-write head, dictate how the machine behaves. Each instruction tells the machine to do three things: replace the symbol of the current cell (which can be replaced with itself), move the head one cell to the left or one cell to the right and move from the current state to a new one (which can be the current state itself).

Initially, the machine's tape contains only an *input string*, while all the other infinite cells contain the blank character. At the end of the computation, the tape contains only the *output string*, which is the result of the computation.