Computational Complexity

Lecture 1: Introduction

Ronald de Haan
me@ronalddehaan.eu

University of Amsterdam

February 2, 2026

Overview of practical information

- Lecturer: Ronald de Haan (`me@ronalddehaan.eu`)
- TA: Arie Soeteman (`a.w.soeteman@uva.nl`)
- Course web page: `https://staff.science.uva.nl/r.dehaan/complexity2026/`
- Canvas page: `https://canvas.uva.nl/courses/56615`
- Book: *Computational Complexity: A Modern Approach* (Arora & Barak, 2009)

What will we do today?

- Getting to know each other a bit
- Some explanations about the course and the topic
- Practical things about the course

- Fundamentals of computational complexity:
  Turing machines, big O notation, decision problems, the complexity class P

What is Computational Complexity?

- The study of what you can compute with limited resources
  - E.g.: time, memory space, random bits
    but also: nondeterminism, oracles

- *Computability theory* (or *recursion theory*) studies what can be computed **in principle**

- *Computational complexity theory* studies what can be computed **realistically**

What is Computational Complexity? (ct'd)

- Main methodology: distinguish different degrees of difficulty (complexity classes)
  - There is an entire 'zoo' of complexity classes:
    `https://www.complexityzoo.net/`
    (currently listing 550 classes)

- One central question: the **P versus NP problem**
  (one of the $1M *Millennium Prize Problems*)

Relation to other fields

(Or in other words: a bit of marketing)

- Computation plays a role in many areas of society and science
- Therefore, computational complexity is relevant for many areas, e.g.:
  - Computer science, cryptography
  - Economics, game theory
  - Artificial intelligence
  - Biology
  - etc.

Course activities

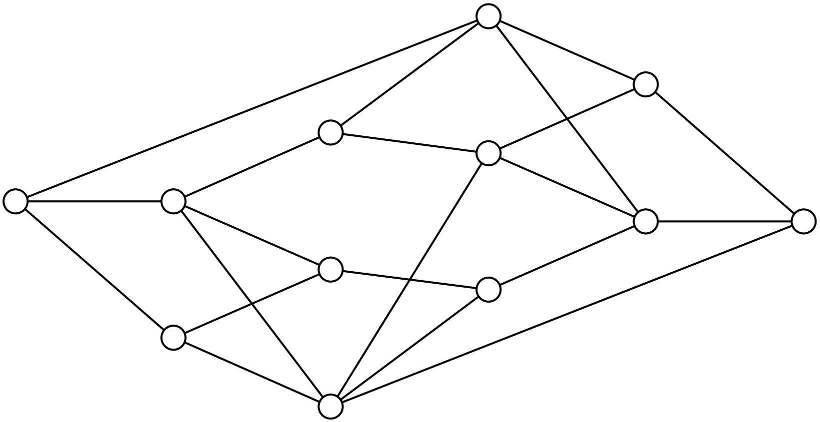
- Lectures:
  - Twice 45 minutes, with 15 minute break in between, not recorded
- Exercise sessions:
  - Practice with material, discuss previous homework assignments
- Homework assignments (50% of grade):
  - Four assignments, hand in via Canvas
- Take-home exam (50% of grade):
  - At the end, open book, one week time to complete exam

Interactive exercise: graph coloring

- You are given an undirected graph
- The task is to color each node with one of k colors so that **no two connected nodes have the same color**
- *Example application:* nodes are regions with their own radio station, colors are radio frequencies, and two nodes are connected if the regions border each other; assign radio frequencies without conflict (in the border areas)

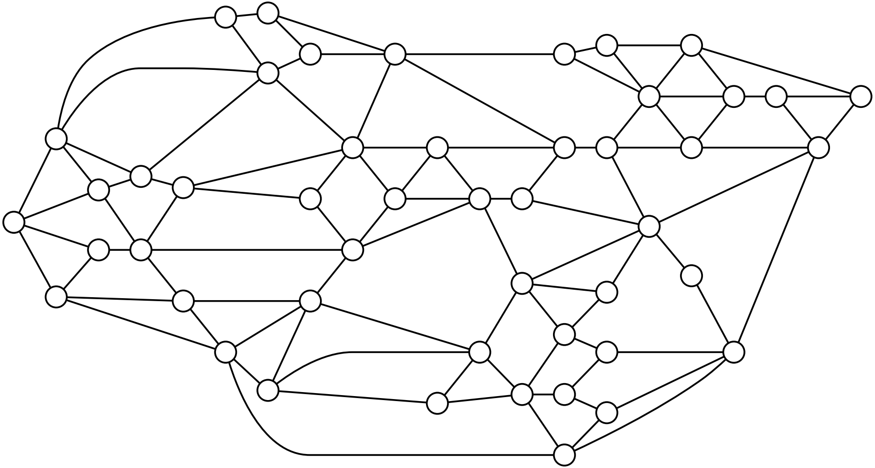
Color this graph with 2 colors

https://tiny.cc/2col



Color this graph with 3 colors

https://tiny.cc/3col



Quadratic vs. Exponential

- Important difference between
  algorithms that run in time, say, n^2
  vs. algorithms that run in time, say, 2^n

- Illustration (time needed for 10^{10} steps per second):

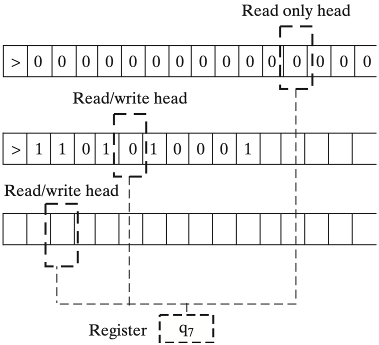
n	n^2 steps	2^n steps
2	0.00000002 msec	0.00000002 msec
5	0.00000015 msec	0.00000019 msec
10	0.00001 msec	0.0001 msec
20	0.00004 msec	0.10 msec
50	0.00025 msec	31.3 hours
100	0.001 msec	9.4×10^{11} years
1000	0.100 msec	7.9×10^{282} years

- # of atoms in universe $\approx 10^{80}$

Definition (Turing machines; TMs)

A *Turing machine* $\mathbb{M}$ is a tuple (Γ, Q, δ), where:

- Γ is the *alphabet*: a finite set of symbols, including 0, 1, $\square$ (the blank symbol), and $\triangleright$ (the start symbol)
- Q is a finite set of *states*, including a designated start state q_{start} and a designated halting state q_{halt}
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathsf{L}, \mathsf{R}, \mathsf{S}\}^k$ is a *transition function*, for some $k \geq 2$ (the number of tapes of the machine)



Model of computation (ct'd)

Definition (TM computing a function)

A TM $\mathbb{M}$ computes the following (partial) function f, where for each $x \in \Sigma^*$:

- $f(x) = y$ if $\mathbb{M}$ halts on input x with output y,
- $f(x) =$ undefined if $\mathbb{M}$ does not halt on input x

Definition (running time)

Let $\mathbb{M}$ be a TM and $g : \mathbb{N} \to \mathbb{N}$ be a function. Then $\mathbb{M}$ *runs in time* $g(n)$ if for each input $x \in \Sigma^n$ of length n, the machine $\mathbb{M}$ halts after (at most) $g(n)$ steps.

- **Note:** we will switch (often implicitly) between
  the conceptual level ("algorithms")
  and the fully formal level ("Turing machines")

- Typically, we are interested in how (roughly) the running time scales, not in all the details
- We use what is called asymptotic analysis

Definition (Big O)

Let $f, g : \mathbb{N} \to \mathbb{N}$. We say that f is $O(g)$ if there exists a constant $c \in \mathbb{N}$ and an $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

- **Note:** in addition to "f is $O(g)$", the following are also used: "$f = O(g)$", "$f \in O(g)$", "$f(n)$ is $O(g(n))$", etc.

For example,
$4n^2 + 3n + 10$ is $O(n^2)$

Take $c = 8$
and $n_0 = 4$

Decision problems

- To simplify the theory, we restrict our attention to yes/no questions

Definition (Decision problems)

A *decision problem* is a function $f : \Sigma^* \to \{0, 1\}$ where for each input $x \in \Sigma^*$ the correct output $f(x)$ is either 0 or 1.

Alternatively: a formal language $L \subseteq \Sigma^*$ where $x \in L$ if and only if $f(x) = 1$.

- For decision problems, we typically look at TMs that have two halting states: q_{acc} (for *accept*: $f(x) = 1$) and q_{rej} (for *reject*: $f(x) = 0$)

The complexity class P

Definition (polynomial-time computability)

A function $f : \Sigma^* \to \Sigma^*$ is *polynomial-time computable* (or *computable in polynomial time*)
if there exist a TM $\mathbb{M}$ and a constant $c \in \mathbb{N}$ such that:

- $\mathbb{M}$ computes f
- $\mathbb{M}$ runs in time $O(|x|^c)$

Definition (the complexity class P)

P is the class (set) consisting of all decision problems $L \subseteq \Sigma^*$ that are computable in polynomial time.

Example algorithm description

- Example of (the main points) in the description of a polynomial-time algorithm for 2-colorability:
  - Pick an arbitrary node, and color it with an arbitrary color.
  - Repeat the following, until either the entire graph is colored, or until two adjacent nodes have the same color.
    - Whenever there is an uncolored node n_1 that is adjacent to a colored node n_2, color n_1 with the opposite color of n_2.
  - If two adjacent nodes have the same color, return "no." If the entire graph is colored and no two adjacent nodes have the same color, return "yes."

Recap

- 2-coloring vs. 3-coloring
- n^2 vs. 2^n
- Turing machines
- Decision problems
- Polynomial time and the class P

Next time

- The universal Turing machine
- Nondeterministic Turing machines
- More complexity classes: NP and coNP
- Polynomial-time reductions