CS 517: Computational Complexity

prof = Mike Rosulek <rosulekm@eecs>

TA = Ni Trieu <trieun@oregonstate>

website = Canvas

textbook = online lecture notes

HW 1 out, due next Fri

Overview

- What is computational complexity about?
- Prerequisites
- *“I heard this class is impossible*”
- Diagonalization

What is computational complexity about?

What complexity is about

- What problems can/cannot be computed with given **resources**?
  - Resource = time, memory, randomness, nondeterminism, interaction…

- How much do different resources help?
  - Is memory more valuable than time? Is randomness more valuable than nondeterminism?

What complexity is about

- What makes some computational problems harder than others?

- How can I {recognize, prove} that a particular problem can’t be solved efficiently?

Prerequisites

I will assume you know ...

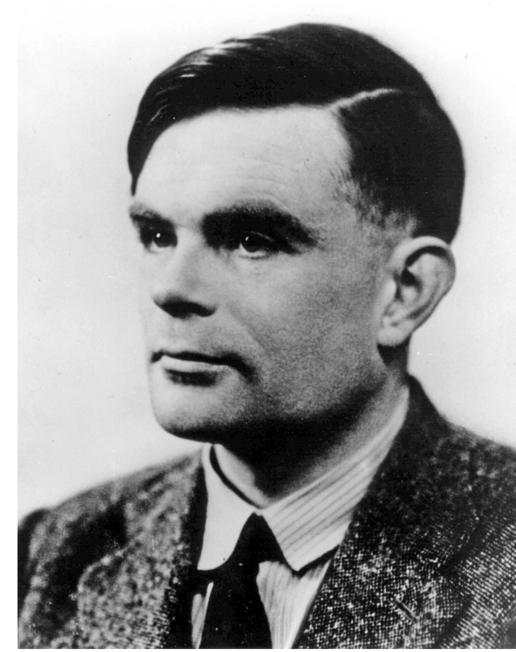
- How to make a logical mathematical argument
  - Induction, contrapositive, etc
- If not:
  - ??
- Goal: your mathematical communication skills (precision, coherence) will improve in this course

I will assume you know ...

- Classic undergrad algorithms
  - Dynamic programming
  - Graph search algorithms
  - Understanding resource usage of an algorithm
- If not:
  - Take cs325 or cs515 first

I will assume you know ...

- Basics of **Turing machines**
  - Definitions, concepts
  - “languages” (decision problems)



- If not:
  - Read Erickson section 6 (TMs) for refresher.
  - If you've literally never seen anything like this, then we should talk.

(Aside: Why Turing machines?)

- Need *some* mathematical definition to formally reason about computation in general
- Historical importance
- Very low-level model (only a few basic operations)
  - Makes **reasoning about computation** easier
  - Makes programming harder (not our concern in this course)

- In the end, choice isn't crucial anyway (any “reasonable” model will be equivalent)

I will assume you know ...

- That Turing machines are **robust to changes** in the computational model
  - In most cases, adding features to a TM doesn't change what problems can be solved
  - e.g.: adding tapes, two-way infinite tapes, etc.

- If not:
  - Read Erickson section 6.6 (TMs)

I will assume you know ...

- That **universal Turing machines** exist
  - A TM's “source code” can be encoded as a string and given to another TM as input
  - A universal TM takes input (M,x) and runs TM M on input x
  - Modern analogy: *you can write a C++ compiler in C++*

ok to say "simulate M on input x"
even if M, x determined @ runtime

- If not:
  - Read Erickson section 6.8, 7.1 (TMs, universal models)

“I heard this class was impossible”

My perspective on a hard course

- My goal is for all CS PhD students to benefit from this course
- Achieving this goal is still a work in progress
- Every CS PhD can benefit from:
  - Understanding what problems can/can't be computed with certain resource constraints
  - Improving their skills in communicating mathematical (especially **algorithmic**) ideas

Course structure

- Lectures, reading
- Homework problem sets (maybe 1 problem per lecture)
- Final exam
  - Likely individualized
  - Subset of homework problems

Diagonalization method

Cardinality

1-to-1 & onto

- **Definition:** |A|=|B| if there is a bijection f:A → B
  - Even if A,B are infinite sets
  - "A & B have same **cardinality**" (there are other useful measures of "size")

rationals

- These sets all have same cardinality:
  - $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\boxed{\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}}$
  - $\mathbb{R}$, $P(\mathbb{N})$

Ex: $\mathbb{N}$ and $\{0, 2, 4, \ldots\}$, bijection $f : \mathbb{N} \to \{0, 2, \ldots\}$

$f(x) = 2x$

Cantor's theorem

- **Theorem**: $|X| \neq |P(X)|$ for every set X (even infinite)
  - **Corollary:** $|\mathbb{Q}| = \boxed{|\mathbb{N}| \neq |P(\mathbb{N})|} = |\mathbb{R}|$
  - *"there are 'more' real numbers than naturals/rationals"*

- Suffices to show:
  - There is no surjective (onto) function $f : X \rightarrow P(X)$
  - Every f always "misses" at least one item in P(X)

edge case: $X = \emptyset$: $|X| = 0$

$P(X) = \{\emptyset\}$ $|P(X)| = 1$

Cantor’s theorem

- **Theorem**: no surjective function f : $\mathbb{X} \rightarrow P(\mathbb{X})$
- **One-line proof:** $\{ x \in \mathbb{X} \mid x \notin f(x) \}$ in $P(\mathbb{X})$ but is never the output of f

Cantor's theorem

- **Theorem**: no surjective function f : X → P(X)

Idea: take $f : X \to P(X)$,

each $f(x) \in P(X) \Rightarrow f(x)$ is subset of X

write f as table

$f(x_1) = \{x_2, x_3 \dots\}$

$f(x_2) = \emptyset$

$f(x_3) = \{x_1, \dots\}$

$$
\begin{array}{c|ccccc}
 & x_1 & x_2 & x_3 & \cdots & x_j \\
\hline
x_1 & 0 & 1 & 1 & & \\
x_2 & 0 & 0 & 0 & & \\
x_3 & 1 & 0 & 0 & & \\
\vdots & & & & & \\
x_i & & & & &
\end{array}
$$

is $x_j \in f(x_i)$?

Cantor's theorem

- **Theorem**: no surjective function f : X → P(X)

	x_1	x_2	x_3	...
x_1	0	1	1	
x_2	0	0	0	.
x_3	1	0	0	
⋮				1

flip

1 1 1 0 ...

Obs: f is onto if every sequence of 0/1 appears somewhere as a row in this table.

Claim: complemented diagonal disagrees w/ every row of table

$\Rightarrow$ disagrees with n^{th} row in (at least) n^{th} position

Diagonalization as a **technique**

- **Goal:** construct a set D that disagrees with all candidates in some list
  - e.g., all possible outputs of f
- **How?**
  - Disagree with 1st candidate on whether 1st item is in/out of the set
  - Disagree with 2nd candidate on whether 2nd item is in/out
  - …