

CS151 Complexity Theory

Lecture 1
March 30, 2021

Complexity Theory

Classify problems according to the **computational resources** required

- running time
- storage space
- parallelism
- randomness
- rounds of interaction, communication, others...

Attempt to answer: what is **computationally feasible** with **limited resources**?

March 30, 2021

CS151 Lecture 1

2

Complexity Theory

- Contrast with decidability: What is computable?
 - answer: some things are not
- We care about resources!
 - leads to many more subtle questions
 - fundamental open problems

March 30, 2021

CS151 Lecture 1

3

The central questions

- Is **finding** a solution as easy as **recognizing** one?
 $P = NP?$
- Is every efficient sequential algorithm **parallelizable**?
 $P = NC?$
- Can every efficient algorithm be converted into one that uses a **tiny amount of memory**?
 $P = L?$
- Are there **small Boolean circuits** for all problems that **require** exponential running time?
 $EXP \in P/poly?$
- Can every efficient randomized algorithm be converted into a **deterministic** algorithm one?
 $P = BPP?$

March 30, 2021

CS151 Lecture 1

4

Central Questions

We *think* we know the answers to all of these questions ...

... **but** no one has been able to prove that even a small part of this “world-view” is correct.

If we're wrong on any one of these then computer science will change dramatically

March 30, 2021

CS151 Lecture 1

5

Introduction

- You already know about two complexity classes
 - **P** = the set of problems decidable in *polynomial time*
 - **NP** = the set of problems with witnesses that can be checked in polynomial time
- ... and notion of **NP-completeness**
- Useful **tool**
- Deep **mathematical problem**: $P = NP?$
 - Course should be **both** useful **and** mathematically interesting

March 30, 2021

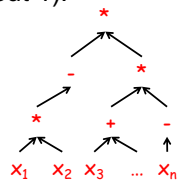
CS151 Lecture 1

6

A question

- Given: polynomial $f(x_1, x_2, \dots, x_n)$ as arithmetic formula (fan-out 1):

- multiplication (fan-in 2)
- addition (fan-in 2)
- negation (fan-in 1)



- Question:** is f identically zero?
(variables take values in finite field of size $>$ degree)

March 30, 2021

CS151 Lecture 1

7

A question

- Given:** multivariate polynomial

$$f(x_1, x_2, \dots, x_n)$$

as an arithmetic formula.

- Question:** is f identically zero?

- Challenge:** devise a deterministic poly-time algorithm for this problem.

March 30, 2021

CS151 Lecture 1

8

A randomized algorithm

- Given:** multivariate degree r poly. $f(x_1, x_2, \dots, x_d)$
note: $r = \deg(f) \leq$ size of formula

- Algorithm:**

- pick small number of **random** points
- if f is zero on all of these points, answer “yes”
- otherwise answer “no”

(low-degree non-zero polynomial evaluates to zero on only a small fraction of its domain)

- No efficient deterministic algorithm known**

March 30, 2021

CS151 Lecture 1

9

Derandomization

- Here is a deterministic algorithm that works under the assumption that there exist hard problems, say SAT.
- solve SAT on all instances of length $\log n$

1 1 0 0 1 1 1 0 0 1

- encode using **error-correcting code** (variant of a Reed-Muller code)

1 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1

March 30, 2021

CS151 Lecture 1

10

Derandomization

1 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1

1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 1

- run randomized alg. using these strings in place of random evaluation points
 - if f is zero on all of these points, answer “yes”
 - otherwise answer “no”
- This works. (proof in this course)

March 30, 2021

CS151 Lecture 1

11

Derandomization

This technique works on **any** randomized algorithm.

Gives generic “derandomization” of randomized procedures.

March 30, 2021

CS151 Lecture 1

12

A surprising fact

- Is *finding* a solution as easy as *recognizing* one?
 $P = NP?$ probably FALSE
- Is *every* sequential algorithm *parallelizable*?
 $P = NC?$ probably FALSE
- Can *every* efficient algorithm be converted into one that uses a *tiny amount of memory*?
 $P = L?$ probably FALSE
- Are there *small Boolean circuits* for *all* problems that *require exponential running time*?
 $EXP \subseteq P/poly?$ probably FALSE
- Can *every* randomized algorithm be converted into a *deterministic* algorithm one?
 $P = BPP?$ probably TRUE

March 30, 2021

CS151 Lecture 1

13

Outline

Should be mostly review...

1. Problems and Languages
2. Complexity Classes
3. Turing Machines
4. Reductions
5. Completeness

March 30, 2021

CS151 Lecture 1

14

Problems and Languages

- Need formal notion of “**computational problem**”. Examples:
 - Given graph G , vertices s, t , find the shortest path from s to t
 - Given matrices A and B , compute AB
 - Given an integer, find its prime factors
 - Given a Boolean formula, find a satisfying assignment

March 30, 2021

CS151 Lecture 1

15

Problems and Languages

- One possibility: function from strings to strings

$$f: \Sigma^* \rightarrow \Sigma^*$$

- **function problem**:
given x , compute $f(x)$
- **decision problem**: $f: \Sigma^* \rightarrow \{\text{yes, no}\}$
given x , accept or reject

March 30, 2021

CS151 Lecture 1

16

Problems and Languages

- simplification doesn't give up much:
 - Given an integer n , find its prime factors
 - Given an integer n and an integer k , is there a factor of n that is $< k$?
 - Given a Boolean formula, find a satisfying assignment
 - Given a Boolean formula, is it satisfiable?
- solve function problem efficiently using related decision problem (how?)
- **We will work mostly with decision problems**

March 30, 2021

CS151 Lecture 1

17

Problems and Languages

- **decision problems**: $f: \Sigma^* \rightarrow \{\text{yes, no}\}$
- equivalent notion: **language** $L \subseteq \Sigma^*$
 $L = \text{set of "yes" instances}$
- Examples:
 - set of strings encoding satisfiable formulas
 - set of strings that encode pairs (n, k) for which n has factor $< k$
- decision problem associated with L :
 - Given x , is x in L ?

March 30, 2021

CS151 Lecture 1

18

Problems and Languages

An aside: two encoding issues

1. implicitly assume we've agreed on a way to encode inputs (and outputs) as strings
 - sometimes relevant in fine-grained analysis (e.g. adj. matrix vs. adj. list for graphs)
 - almost never an issue in this class
 - avoid silly encodings: e.g. unary

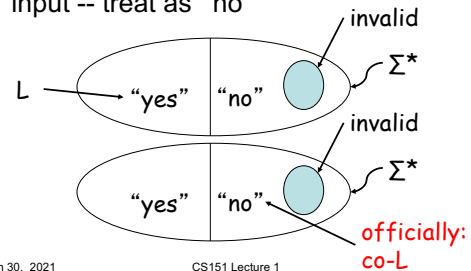
March 30, 2021

CS151 Lecture 1

19

Problems and Languages

2. some strings not valid encodings of any input -- treat as "no"



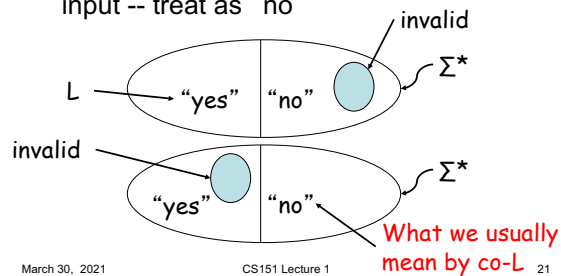
March 30, 2021

CS151 Lecture 1

20

Problems and Languages

2. some strings not valid encodings of any input -- treat as "no"



March 30, 2021

CS151 Lecture 1

21

Complexity Classes

- **complexity class** = class of languages
- set-theoretic definition – no reference to computation (!)
- example:
 - **TALLY** = languages in which every yes instance has form 0^n
 - e.g. $L = \{ 0^n : n \text{ prime} \}$

March 30, 2021

CS151 Lecture 1

22

Complexity Classes

- complexity classes you know:
 - **P** = the set of languages decidable in *polynomial time*
 - **NP** = the set of languages L where

$$L = \{ x : \exists y, |y| \leq |x|^k, (x, y) \in R \}$$
 and R is a language in **P**
- easy to define complexity classes...

March 30, 2021

CS151 Lecture 1

23

Complexity Classes

- ...harder to define **meaningful** complexity classes:
 - **capture** genuine computational phenomenon (e.g. parallelism)
 - **contain** natural and relevant problems
 - ideally **characterized** by natural problems (completeness – more soon)
 - **robust** under variations in model of computation
 - possibly **closed** under operations such as AND, OR, COMPLEMENT...

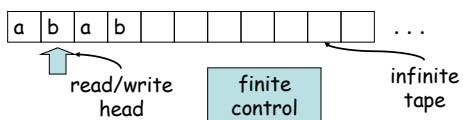
March 30, 2021

CS151 Lecture 1

24

Complexity Classes

- need a **model of computation** to define classes that capture important aspects of computation
- Our model of computation: **Turing Machine**



March 30, 2021

CS151 Lecture 1

25

Turing Machines

- Q finite set of states
- Σ alphabet including blank: “_”
- $q_{start}, q_{accept}, q_{reject}$ in Q
- $\delta: Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, -\}$ transition fn.
- input written on tape, head on 1st square, state q_{start}
- sequence of steps specified by δ
- if reach q_{accept} or q_{reject} then halt

March 30, 2021

CS151 Lecture 1

26

Turing Machines

- three notions of computation with Turing machines. In all, input x written on tape...
 - **function computation**: output $f(x)$ is left on the tape when TM halts
 - **language decision**: TM halts in state q_{accept} if $x \in L$; TM halts in state q_{reject} if $x \notin L$.
 - **language recognition**: TM halts in state q_{accept} if $x \in L$; may loop forever otherwise.

March 30, 2021

CS151 Lecture 1

27

Example:

q	σ	$\delta(q, \sigma)$
start	0	(start, 0, R)
start	1	(start, 1, R)
start	_	(t, _, L)
start	#	(start, #, R)

q	σ	$\delta(q, \sigma)$
t	0	(accept, 1, -)
t	1	(t, 0, L)
t	#	(accept, #, R)

#	0	1								start
#	0	1								start
#	0	1								start
#	0	1								start
#	0	1								t
#	0	0								t
#	1	0								accept

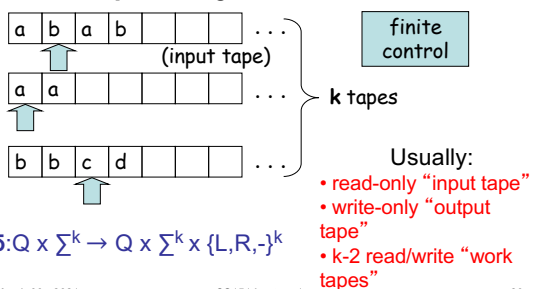
March 30, 2021

CS151 Lecture 1

28

Turing Machines

- **multi-tape Turing Machine**:



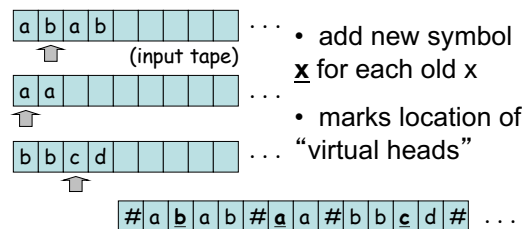
March 30, 2021

CS151 Lecture 1

29

Multitape TMs

simulation of k-tape TM by single-tape TM:

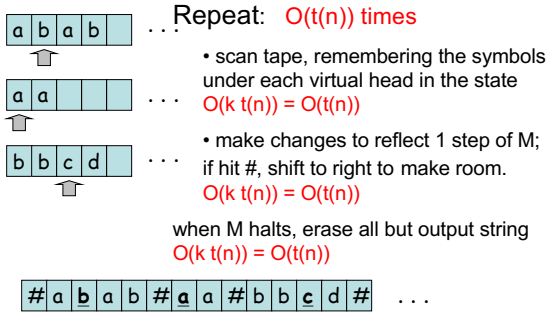


March 30, 2021

CS151 Lecture 1

30

Multitape TMs



March 30, 2021

CS151 Lecture 1

31

Extended Church-Turing Thesis

- the belief that TMs formalize our intuitive notion of an efficient algorithm is:

The “extended” Church-Turing Thesis

everything we can compute in time $t(n)$ on a physical computer can be computed on a Turing Machine in time $t^{O(1)}(n)$ (polynomial slowdown)

- quantum computers challenge this belief

March 30, 2021

CS151 Lecture 1

32

Extended Church-Turing Thesis

- consequence of extended Church-Turing Thesis: all reasonable physically realizable models of computation can be *efficiently* simulated by a TM
- e.g. multi-tape vs. single tape TM
- e.g. RAM model

March 30, 2021

CS151 Lecture 1

33