# BU CS 332 – Theory of Computation

## Lecture 20:

- Time/Space Hierarchies
- Complexity Class P

Reading:

Sipser Ch 9.1, 7.2

Mark Bun

April 7, 2021

# Last Time

- Asymptotic notation
- Analyzing time / space usage of Turing machines (algorithms)
- Multi-tape TMs can solve problems faster than single-tape TMs

# Time complexity

Time complexity of a TM (algorithm) = maximum number of steps it takes on a worst-case input

Formally: Let $f : \mathbb{N} \rightarrow \mathbb{N}$. A TM $M$ runs in time $f(n)$ if on every input $w \in \Sigma^n$, $M$ halts on $w$ within at most $f(n)$ steps

A language $A \in \mathrm{TIME}(f(n))$ if there exists a basic single-tape (deterministic) TM $M$ that

1) Decides $A$, and
2) Runs in time $O(f(n))$

# Single vs. Multi-Tape

Theorem: Let $t(n) \geq n$ be a function. Every multi-tape TM running in time $t(n)$ has an equivalent single-tape TM running in time $O(t(n)^2)$

Proof idea:

We already saw how to simulate a multi-tape TM with a single-tape TM

Need a runtime analysis of this construction

# Applying the simulation theorem

Suppose $B$ is decidable in time $O(n^2)$ on a 42-tape TM. Then a basic single-tape TM can decide $B$ in time…

a) $O(n^2)$
b) $O(n^4)$
c) $O(n^{84})$
d) $2^{O(n)}$

# Simulating Multiple Tapes

(Implementation-Level Description)

On input $w = w_1 w_2 \ldots w_n$

1. Format tape into $\# \dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$
2. For each move of $M$:

    Scan left-to-right, finding current symbols

    Scan left-to-right, writing new symbols,

    Scan left-to-right, moving each tape head

    If a tape head goes off the right end, insert blank

    If a tape head goes off left end, move back right

# Single vs. Multi-Tape

Theorem: Let $t(n) \geq n$ be a function. Every multi-tape TM running in time $t(n)$ has an equivalent single-tape TM running in time $O(t(n)^2)$

Proof: Time analysis of simulation

- Time to initialize (i.e., format tape): $O(n + k)$
- Time to simulate one step of multi-tape TM: $O\big(k \cdot t(n)\big)$

- Number of steps to simulate: $t(n)$

$\Rightarrow$ Total time:

# Extended Church-Turing Thesis

Every "reasonable" (physically realizable) model of computation can be simulated by a basic, single-tape TM with only a **polynomial** slowdown.

E.g., doubly infinite TMs, multi-tape TMs, RAM TMs

Does not include nondeterministic TMs (not reasonable)

Possible counterexamples? Randomized computation, parallel computation, DNA computing, quantum computation

# Space complexity

Space complexity of a TM (algorithm) = maximum number of tape cells it uses on a worst-case input

Formally: Let $f : \mathbb{N} \rightarrow \mathbb{N}$. A TM $M$ runs in space $f(n)$ if on every input $w \in \Sigma^n$, $M$ halts on $w$ using at most $f(n)$ cells

A language $A \in \mathrm{SPACE}(f(n))$ if there exists a basic single-tape (deterministic) TM $M$ that

1) Decides $A$, and
2) Runs in time $O(f(n))$

# How does space relate to time?

Which of the following is true for every function $f(n) \geq n$?

a) $TIME\big(f(n)\big) \subseteq SPACE(f(n))$

b) $SPACE\big(f(n)\big) \subseteq TIME(f(n))$

c) $TIME\big(f(n)\big) = SPACE(f(n))$

d) None of the above

# Back to our examples

$$A = \{0^m 1^m \mid m \geq 0\}$$

Theorem: Let $s(n) \geq n$ be a function. Every multi-tape TM running in time $s(n)$ has an equivalent single-tape TM running in time $O(s(n))$

# Hierarchy Theorems

# More time, more problems

We know, e.g., that $TIME(n^2) \subseteq TIME(n^3)$

(Anything we can do in quadratic time we can do in cubic time)

Question: Are there problems that we can solve in cubic time that we cannot solve in quadratic time?

Theorem: There is a language $L \in TIME(n^3)$,

but $L \notin TIME(n^2)$

"Time hierarchy":

$TIME(n) \subsetneq TIME(n^2) \subsetneq TIME(n^3) \subsetneq TIME(n^4)$ ...

# Diagonalization redux

| TM $M$ | $M(\langle M_1 \rangle)$? | $M(\langle M_2 \rangle)$? | $M(\langle M_3 \rangle)$? | $M(\langle M_4 \rangle)$? | | $D(\langle D \rangle)$? |
|---|---|---|---|---|---|---|
| $M_1$ | Y | N | Y | Y | … | |
| $M_2$ | N | N | Y | Y | | |
| $M_3$ | Y | Y | Y | N | | |
| $M_4$ | N | N | Y | N | | |
| ⋮ | | | | | ⋱ | |
| $D$ | | | | | | |

$$UD = \{\langle M \rangle \mid M \text{ is a TM that does not accept input } \langle M \rangle\}$$

$$L = \{\langle M \rangle \mid M \text{ is a TM that does not accept input } \langle M \rangle \text{ within } n^{2.5} \text{ steps}\}$$

# An explicit separating language

Theorem: $L = \{\langle M \rangle \mid M \text{ is a TM that does not accept input } \langle M \rangle \text{ within } n^{2.5} \text{ steps}\}$

is in $TIME(n^3)$, but not in $TIME(n^2)$

Proof: In $TIME(n^3)$

On input $\langle M \rangle$:

1. Simulate $M$ on input $\langle M \rangle$ for $n^{2.5}$ steps
2. If $M$ accepts, reject. If $M$ rejects or did not yet halt, accept.

# An explicit separating language

Theorem: $L = \{\langle M \rangle \mid M \text{ is a TM that does not accept input } \langle M \rangle \text{ within } n^{2.5} \text{ steps}\}$

is in $TIME(n^3)$, but not in $TIME(n^2)$

Proof: Not in $TIME(n^2)$

Suppose for contradiction that $D$ decides $L$ in time $O(n^2)$

# Time and space hierarchy theorems

- For any* function $t(n) \geq n \log n$, a language exists that is decidable in $t(n)$ time, but not in $o\left(\frac{t(n)}{\log t(n)}\right)$ time.

- For any* function $s(n) \geq \log n$, a language exists that is decidable in $s(n)$ space, but not in $o(s(n))$ space.

*"time constructible" and "space constructible", respectively

# Complexity Class $\mathbf{P}$

# Time and space complexity

The basic questions

1. How do we measure complexity?

2. Asymptotic notation

3. How robust is the TM model when we care about measuring complexity?

4. How do we mathematically capture our intuitive notion of “efficient algorithms”?

# Complexity class P

Definition: P is the class of languages decidable in polynomial time on a basic single-tape (deterministic) TM

$$\mathrm{P} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(n^k)$$

- Class doesn't change if we substitute in another reasonable deterministic model (Extended Church-Turing)
- Cobham-Edmonds Thesis: Roughly captures class of problems that are feasible to solve on computers

# Check your type checker: P

Consider the following computational problem: Given two numbers $x, y$ (written in binary), output their sum $x + y$ (in binary). Which of the following is true?

a) This is a problem in P
b) This problem is not in P because it cannot be solved by a Turing machine (i.e., it's undecidable)
c) This problem is not in P because it cannot be solved in polynomial time
d) This problem is not in P because it is not a decision problem (i.e., a language)

# A note about encodings

We'll still use the notation $\langle \quad \rangle$ for "any reasonable" encoding of the input to a TM…but now we have to be more careful about what we mean by "reasonable"

How long is the encoding of a $V$-vertex, $E$-edge graph…

… as an adjacency matrix?

… as an adjacency list?

How long is the encoding of a natural number $k$

… in binary?

… in decimal?

… in unary?

# When the encoding matters

The "same" algorithm can have wildly different runtimes depending on how the input is encoded!

$M$ = "On input $x$ (a natural number encoded in binary):

1. List the binary numbers $1, 2, \ldots, x$ and accept."

$M'$ = "On input $x$ (a natural number encoded in unary):

1. List the binary numbers $1, 2, \ldots, x$ and accept."

# Describing and analyzing polynomial-time algorithms

- Due to Extended Church-Turing Thesis, we can still use high-level descriptions on multi-tape machines
- Polynomial-time is robust under composition: $\text{poly}(n)$ executions of $\text{poly}(n)$-time subroutines run on $\text{poly}(n)$-size inputs gives an algorithm running in $\text{poly}(n)$ time.

  $\Rightarrow$ Can freely use algorithms we've seen before as subroutines if we've analyzed their runtime

- Need to be careful about size of inputs! (Assume inputs represented in binary unless otherwise stated.)

# Examples of languages in $\mathrm{P}$

$PATH =$
$\{\langle G, s, t\rangle \mid G \text{ is a directed graph with a directed path from } s \text{ to } t\}$

# Examples of languages in $\mathrm{P}$

$$E_{\mathrm{DFA}} = \{\langle D \rangle \mid D \text{ is a DFA that recognizes the empty language}\}$$

# Examples of languages in P

- $RELPRIME = \{\langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime}\}$

- $PRIMES = \{\langle x \rangle \mid x \text{ is prime}\}$

2006 Gödel Prize citation



The 2006 Gödel Prize for outstanding articles in theoretical computer science is awarded to Manindra Agrawal, Neeraj Kayal, and Nitin Saxena for their paper "PRIMES is in P."

In August 2002 one of the most ancient computational problems was finally solved….

# A polynomial-time algorithm for $PRIMES$?

Consider the following algorithm for $PRIMES$

On input $\langle x \rangle$:

For $b = 2, 3, 4, 5, \ldots, \sqrt{x}$:

- Try to divide $x$ by $b$
- If $b$ divides $x$, accept

If all $b$ fail to divide $x$, reject

How many divisions does this algorithm require in terms of $n = |\langle x \rangle|$? a) $O(\sqrt{n})$ b) $O(n)$ c) $2^{O(\sqrt{n})}$ d) $2^{O(n)}$

# Beyond polynomial time

Definition: EXP is the class of languages decidable in exponential time on a basic single-tape (deterministic) TM

$$\mathrm{EXP} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(2^{n^k})$$

# Why study P ?

Criticism of the Cobham-Edmonds Thesis:

- Algorithms running in time $n^{100}$ aren't really efficient

  Response: Runtimes improve with more research

- Does not capture some physically realizable models using randomness, quantum mechanics

  Response: Randomness may not change P, useful principles

$TIME(n)$ vs. $TIME(n^2)$

$P$ vs. $EXP$

decidable vs. undecidable