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...

hidden in O after simulation

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

all of these are true

$f(n) = 2^{O(n)} \iff \exists\ g(n) = O(n)$ s.t. $f(n) = 2^{g(n)}$

Ex: $4^n = 2^{O(n)}$ [since $4^n = 2^{2n}$ and $2n = O(n)$]

but $4^n \neq O(2^n)$

Simulating Multiple Tapes

(Implementation-Level Description)

On input $w = w_1 w_2 \ldots w_n$

1. Format tape into $\# \dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$ ← $O(n+k)$ time
   ($O(n)$ time; $O(k)$ time)
2. For each move of M: ($t(n)$ moves to simulate)
   - 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

   Each move takes $O(k \cdot t(n))$ time

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: $O(k \cdot t(n))$

Why? Each simulated tape consists of $\leq t(n)$ cells, since M runs in time (and hence space) $t(n)$
$\Rightarrow$ scanning tape takes time $O(k \cdot t(n))$

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

$\Rightarrow$ Total time: $\underbrace{O(n+k)}_{\text{step 1}} + \underbrace{t(n)}_{\#\text{ moves}} \cdot \underbrace{O(k \cdot t(n))}_{\text{time per move}} = O(k \cdot (t(n))^2)$

(constant indep. of n)

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

More believable: Restrict ECT to deterministic, sequential models of 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 space $O(f(n))$

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

False
Hopcroft - Paul - Valiant '77

M runs in time f(n)
⇒ M touches ≤ f(n) cells on every input of length n
⇒ M runs in space f(n)
TIME(f(n)) ⊆ SPACE(f(n))

Back to our examples

Linear bounded automaton

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

$\in$ SPACE (n)

M = "On input x:

1) Check $x \in L(0^* 1^*)$
2) Repeatedly cross off one 0 and one 1 until impossible
3) Accept iff everything crossed off

O(n) space algorithm

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

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	N	Y	Y	…	
M_2	N	N Y	Y	Y		
M_3	Y	Y	Y N	N		
M_4	N	N	Y	N Y		
⋮					⋱	
D						

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

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

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 $\langle M \rangle$:

Argue that UTM can simulate M(<M>) w/ low "overhead" => $O(n^3)$ time algorithm

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 $TIME(n^2)$

simplifying assumption that n^2 time

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

D accepts input $\langle D \rangle$: D is a TM accepting $\langle D \rangle$ w/in n^2 steps
$\Rightarrow D \notin L$ ✗

D rejects input $\langle D \rangle$: D rejects $\langle D \rangle$ w/in n^2 steps
$\Rightarrow D \in L$ ✗

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.

  Ex: $t(n) = n \log^2 n$

  $\text{TIME}(n \log^2 n) \not\subseteq \text{TIME}(n)$

- 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

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

$$= \mathrm{TIME}(n) \cup \mathrm{TIME}(n^2) \cup \mathrm{TIME}(n^3) \cup \ldots$$

- 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

$L = \{\langle x, y, x+y \rangle \mid x, y \in \mathbb{N}\}$
$L \in P$

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?

P is a set of languages

language L is a set of strings

$x \in L$ if answer to instance x is "yes"
$x \notin L$ if answer to instance x is "no"

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 ⟨ ⟩ 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? $O(V^2)$

… as an adjacency list? $O(E+V)$

Encoding doesn't really matter for measuring runtime up to polynomials

How long is the encoding of a natural number k

… in binary? $n = \log_2 k$

… in decimal? $n = \log_{10} k$

> same up to constant

… in unary? $n = k$ ← drastic difference

11111 (n)

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."