BU CS 332 – Theory of Computation

Lecture 18:

- Time Complexity
- Complexity Class P

Reading:

Sipser Ch 7.1-7.2

Ran Canetti

November 12, 2020

Where we are in CS 332

Automata & Formal Languages	Computability	Complexity

Previous unit: Computability theory:
What kinds of problems can / can’t computers solve?

Final unit: Complexity theory:
How much time/memory does it take to solve a problem?
What problems can computers solve with bounded resources?

➔ We first concentrate on time complexity

Today

1. How do we measure complexity? (as in CS 330)

2. Asymptotic notation (as in CS 330)

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”?

Analyzing Runtime

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

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

- Focus on worst-case running time: Upper bound of $f(n)$ must hold for all inputs of length n
- Exact running time $f(n)$ does not translate well between computational models / real computers. Instead focus on asymptotic complexity.

Example

How much time does it take for a basic single-tape TM to decide $A = \{0^m 1^m \mid m \geq 0\}$?

Let's analyze one particular TM M:

```
$M$ = "On input $w$:
        1. Scan input and reject if not of the form $0^*1^*$
        2. While input contains both $0$'s and $1$'s:
                Cross off one $0$ and one $1$
        3. Accept if no 0's and no 1's left. Otherwise, reject."
```

Review of asymptotic notation

O-notation (upper bounds)

$f(n) = O(g(n))$ means:

There exist constants $c > 0, n_0 > 0$ such that $f(n) \leq cg(n)$ for every $n \geq n_0$

Example: $2n^2 = O(n^3)$ $(c = 2, n_0 = 0)$

Caution: $=$ does not mean "equals"

Not reflexive:

$f(n) = O(g(n))$ does not mean $g(n) = O(f(n))$

Example: $f(n) = 2n^2$, $g(n) = n^3$

Alternative (better) notation: $f(n) \in O(g(n))$

Examples

- $10^6 n^3 + 2n^2 - n + 10 =$
- $\sqrt{n} + \log n =$
- $n(\log n + \sqrt{n}) =$
- $n =$

Review of asymptotic notation

Ω-notation (lower bounds)

$f(n) = \Omega(g(n))$ means:

There exist constants $c > 0, n_0 > 0$ such that $f(n) \geq cg(n)$ for every $n \geq n_0$

Example: $\sqrt{n} = \Omega(\log n)$ $(c = 1, n_0 = 16)$

When should we use O vs. Ω?

Upper bounds: Use O

“The merge-sort algorithm uses at most $O(n \log n)$ comparisons in the worst case”

Lower bounds: Use Ω

“Every comparison-based sorting algorithm requires at least $\Omega(n \log n)$ comparisons in the worst case”

Review of asymptotic notation

Θ-notation (tight bounds)

$f(n) = \Theta(g(n))$ means:

$$f(n) = O(g(n)) \quad \text{AND} \quad f(n) = \Omega(g(n))$$

Example: $\frac{1}{2}n^2 - 1000n = \Theta(n^2)$

Generally, polynomials are easy:

$$a_d n^d + a_{d-1}n^{d-1} + \cdots + a_1 n + a_0 = \Theta(n^d)$$

Little-oh and little-omega

O-notation and Ω-notation are like $\leq$ and $\geq$;
o-notation and ω-notation are like $<$ and $>$

$f(n) = o(g(n))$ means:
For every constant $c > 0$, there exists $n_0 > 0$ such that
$f(n) \leq cg(n)$ for every $n \geq n_0$

Example: $2n^2 = O(n^3)$ $(n_0 = 2/c)$

A handy-dandy chart

Notation	… means …	Think…	Example	$\lim\limits_{n\leftarrow\infty} \frac{f(n)}{g(n)}$
$f(n)=O(g(n))$	$\exists\ c>0, n_0>0, \forall\ n > n_0$: $f(n) < cg(n)$	Upper bound	$100n^2$ $= O(n^3)$	If it exists, it is $< \infty$
$f(n)=\Omega(g(n))$	$\exists c>0, n_0>0, \forall\ n > n_0$: $cg(n) < f(n)$	Lower bound	2^n $= \Omega(n^{100})$	If it exists, it is > 0
$f(n)=\Theta(g(n))$	both of the above: $f=\Omega(g)$ **and** $f = O(g)$	Tight bound	$\log(n!)$ $= \Theta(n \log n)$	If it exists, it is > 0 and $< \infty$
$f(n)=o(g(n))$	$\forall\ c>0, \exists\ n_0>0, \forall\ n > n_0$: $f(n) < cg(n)$	Strict upper bound	$n^2 = o(2^n)$	Limit exists, $=0$
$f(n)=\omega(g(n))$	$\forall\ c>0, \exists n_0>0, \forall\ n > n_0$: $cg(n) < f(n)$	Strict lower bound	n^2 $= \omega(\log n)$	Limit exists, $=\infty$

Asymptotic notation within expressions

Asymptotic notation within an expression is shorthand for an unspecified function satisfying the statement

Examples:

- $n^{O(1)}$
- $n^2 + \Omega(n)$
- $\big(1 + o(1)\big)n$

FAABs: Frequently asked asymptotic bounds

- Polynomials. $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n$
- Logarithms. $\log_a n = \quad (\log_b n)$ for all consta

  For every $c > 0$, $\log n = o(n^c)$

- Exponentials. For all $b > 1$ and all $d > 0$, n
- Factorial. $n! = n(n-1)\cdots 1$

  By Stirling's formula,

$$n! = (\sqrt{2\pi n})\left(\frac{n}{e}\right)^n (1 + o(1)) = 2^{\Theta(n \log n)}$$

Time Complexity

Time complexity classes

Let $f : \mathbb{N} \to \mathbb{N}$

$\mathrm{TIME}(f(n))$ is a class (i.e., set) of languages:

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 $O(f(n))$

Example

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

```
$M$ = "On input $w$:
        1. Scan input and reject if not of the form $0^*1^*$
        2. While input contains both $0$'s and $1$'s:
               Cross off one $0$ and one $1$
        3. Accept if no $0$'s and no $1$'s left. Otherwise, reject."
```

- M runs in time $O(n^2)$

- Is there a faster algorithm?

Example

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

M' = "On input w:

```
1. Scan input and reject if not of the form $0^*1^*$
2. While input contains both $0$'s and $1$'s:
    • Reject if the total number of 0's and 1's remaining is odd
    • Cross off every other $0$ and every other $1$
3. Accept if no 0's and no 1's left. Otherwise, reject."
```

- Running time of M':

- Is there a faster algorithm?

Example

Running time of M': $O(n \log n)$

Theorem (Sipser, Problem 7.49): If L can be decided in $o(n \log n)$ time on a 1-tape TM, then L is regular

Does it matter that we're using the 1-tape model for this result?

It matters: 2-tape TMs can decide A faster

```
$M''$ = "On input $w$:
    1. Scan input and reject if not of the form $0^*1^*$
    2. Copy $0$'s to tape 2
    3. Scan tape $1$. For each $1$ read, cross of a 0 on tape 2
    4. If $0$'s on tape 2 finish at same time as $1$'s on tape 1, accept.
       Otherwise, reject."
```

Analysis: A is decided in time $O(n)$ on a 2-tape TM

Moral of the story (part 1): Unlike decidability, time complexity depends on the TM model

How *much* does the model matter?

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

Moral of the story (part 2): Time complexity doesn't depend too much on the TM model (as long as it's deterministic, sequential)

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

How *much* does the model matter?

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: $O\big(t(n)\big)$

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

=> Total time:

Extended Church-Turing Thesis

Every “reasonable” 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

Complexity Class P

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: Captures class of problems that are feasible to solve on computers

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 an n-vertex graph…

… as an adjacency matrix?

… as an adjacency list?

How long is the encoding of a natural number n

… in binary?

… in decimal?

… in unary?

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: $\mathrm{poly}(n)$ executions of $\mathrm{poly}(n)$-time subroutines run on $\mathrm{poly}(n)$-size inputs gives an algorithm running in $\mathrm{poly}(n)$ time.

  => 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 P

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

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

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 $b = 2, 3, 5, \ldots, \sqrt{x}$:
        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 $n = |\langle x \rangle|$?