BU CS 332 – Theory of Computation

https://forms.gle/MabPvkC4K9jNHD257

Lecture 18:

- Asymptotic Notation
- Time/Space Complexity

Reading:

Sipser Ch 7.1, 8.0

Mark Bun

November 11, 2021

Where we are in CS 332

Automata	Computability	Complexity

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

Final unit: Complexity theory
What kinds of problems can / can't computers solve under constraints on their computational resources?

Time and space complexity

Today: Start answering the basic questions

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"?

Time and space complexity

Time complexity of a TM = Running time of an algorithm

= Max number of steps as a function of input length n

Space complexity of a TM = Memory usage of algorithm

= Max number of tape cells as a function of input length n

Example

In how much time/space can a basic single-tape TM 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."

Example

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."

What is the time complexity of M?

a) $O(1)$ [constant time]

b) $O(n)$ [linear time]

c) $O(n^2)$ [quadratic time]

d) $O(n^3)$ [cubic time]

What is the space complexity of M?

Review of asymptotic notation

O-notation (upper bounds)

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

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

Example: $2n^2 + 12 = O(n^3)$ $(c = 3, n_0 = 4)$

Properties of asymptotic notation:

Transitive:

$f(n) = O(g(n))$ and $g(n) = O(h(n))$ means $f(n) = O(h(n))$

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

- $10^6\, n^3 + \ 2n^2 \ - n \ + 10 \ =$

- $\sqrt{n} \ + \ \log n \quad =$

- $n\ (\log n + \sqrt{n}) \ =$

- $n \ =$

Little-oh

If O-notation is like $\leq$, then o-notation is like $<$

$f(n) = o(g(n))$ 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: $2n^2 + 12 = o(n^3)$ $\quad (n_0 = \max\{4/c, 3\})$

True facts about asymptotic expressions

Which of the following statements is true about the function $f(n) = 2^n$?

a) $f(n) = O(3^n)$

b) $f(n) = o(3^n)$

c) $f(n) = O(n^2)$

d) $n^2 = O(f(n))$

Asymptotic notation within expressions

Asymptotic notation within an expression is shorthand for "there exists a function satisfying the statement"

Examples:

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

FAABs: Frequently asked asymptotic bounds

- **Polynomials.** $a_0 + a_1 n + \ldots + a_d n^d$ is $O(n^d)$ if $a_d > 0$
- **Logarithms.** $\log_a n = O(\log_b n)$ for all constants $a, b > 0$

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

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

By Stirling's formula,

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

Time and Space Complexity

Running time analysis

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

- 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.

Time complexity classes

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

$\mathrm{TIME}(f(n))$ is a set ("class") 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 $O(f(n))$

Time class containment

If $f(n) = O(g(n))$, then which of the following statements is always true?

a) $\text{TIME}\big(f(n)\big) \subseteq \text{TIME}\big(g(n)\big)$

b) $\text{TIME}\big(g(n)\big) \subseteq \text{TIME}\big(f(n)\big)$

c) $\text{TIME}\big(f(n)\big) = \text{TIME}\big(g(n)\big)$

d) None of the above

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 off 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)

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

Space complexity

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

Formally: Let $f : \mathbb{N} \to \mathbb{N}$. 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 $O(f(n))$

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 space $s(n)$ has an equivalent single-tape TM running in space $O(s(n))$