Complexity theory: Introduction

Evgenij Thorstensen

V18

Course outline

We will cover chapters 7 and 8 from Sipser.

We will also cover some topics/theorems from chapters 9-10, depending on time constraints.

Probably 9.1 and 9.2, but I will get back to that.

What is complexity theory?

So far, you've studied *models of computation* and their *expressive power*.

Complexity theory deals with measuring the amount of resources required for computational problems.

A typical problem is "deciding a language L".

This idea raises many questions:

- What resources are we interested in?
- What model of computation to use for "algorithm"?
- How do we measure this?

Resources and measuring

Assume a model of computation, e.g. single-tape DTM.

Two very obvious resources of interest: Time and space.

Time: Number of DTM transitions needed.

Space: Number of tape cells needed.

Given a DTM M that decides a language L, and a string w, it now makes sense to talk about the time and space needed.

Generalizing measuring

Given a string w and a DTM M for L, there is a number of transitions n that M needs to accept or reject w.

This number depends on w and on M.

We can generalize by defining functions to measure this over *all* w and even *all* M.

Generalizing measuring

Given a string w and a DTM M for L, there is a number of transitions n that M needs to accept or reject w.

This number depends on w and on M.

We can generalize by defining functions to measure this over *all* w and even *all* M.

Let $f_M(w)$ be the number of transitions M uses for w. We expect this number to depend on $|w|$.

Define $W_k = \{f_M(w) \mid |w| = k\}$, and $t_M(k) = \max(W_k)$.

Time as a function of input size

With $t_M(k) = \max(W_k)$, we get $t_M : \mathbb{N} \to \mathbb{N}$. $t_M(k)$ is the running time of M (Def. 7.1).

This is the worst-case running time.

Allows us to compare different languages using different encodings!

Time as a function of input size

With $t_M(k) = \max(W_k)$, we get $t_M : \mathbb{N} \to \mathbb{N}$. $t_M(k)$ is the running time of M (Def. 7.1).

This is the worst-case running time.

Allows us to compare different languages using different encodings!

Let M_1 and M_2 be deciders for L_1 and L_2. Say we have

$$t_{M_1}(k) = 2k^3$$

and

$$t_{M_2}(k) = 2^k$$

Time as a function of a language

Let L be a language. We can define the set

$$\{t_M \mid M \text{ decides } L\}$$

We would like to speak about the time needed to decide L (by the fastest DTM, usually).

Need to compare functions of n.

Comparing functions

The primary way of comparing functions is by growth rate.

Definition (Big-O, Def 7.2)

Let f and g be functions $f, g : \mathbb{N} \to \mathbb{R}^+$. We say that $f(n) = O(g(n))$ if there exists

- a threshold $n_0 \in \mathbb{N}$ and
- a constant $c \in \mathbb{N}$

such that for *every* $n \geqslant n_0$,

$$f(n) \leqslant c \cdot g(n)$$

Big-Oh, graphically

$\exists n_0, c$ such that $\forall n \geqslant n_0$ we have $f(n) \leqslant c \cdot g(n)$:

Big-Oh, properties

We get to choose c: Constants are ignored.
$300n^3 = O(n^3) = O(100n^3)$

We get to choose n_0: Small-number behaviour is ignored.

Logarithms: Base doesn't matter. $\log_b n = \frac{\log_2 n}{\log_2 b}$, so
$\log_b n = O(\log_d n)$ for any b, d.

Notational abuse: Technically, $O(g(n))$ is a set, so it ought to be
$f \in O(g)$.

Expressions like $2^{O(n)}$, $2^{O(1)}$, and even (arrgh) $2^{O(\log n)}$ happen.

More on arithmetic with O

Expressions like $2^{O(n)}$, $2^{O(1)}$, and even (arrgh) $2^{O(\log n)}$ happen.

$2^{O(\log n)} = 2^{c \log n}$ for some c.

Since $n = 2^{\log_2 n}$, $n^c = 2^{c \log_2 n}$. Therefore $2^{O(\log n)} = n^c = n^{O(1)}$.

Sum rule

Sum rule: If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then $f_1(n) + f_2(n) \in O(\max(g_1(n), g_2(n)))$.

The sum grows with the fastest term; others can be discarded.

Proof: Assume $f_1(n) \leqslant c_1 g_1(n)$ from n_1, and $f_2 \leqslant c_2 g_2(n)$ from n_2.

Apply sums: $f_1(n) + f_2(n) \leqslant c_1 g_1(n) + c_2 g_2(n)$ from $\max(n_1, n_2)$.

It follows that $f_1(n) + f_2(n) \leqslant 2 \cdot \max(c_1, c_2) \cdot \max(g_1(n), g_2(n))$ from $\max(n_1, n_2)$.

$n^3 + n^6 \in O(n^6)$, since $2n^6$ is greater than the sum.

Product rule

Product rule: If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then $f_1(n) \times f_2(n) \in O(g_1(n) \times g_2(n))$.

Product of bounds is a bound on the product.

Proof: Exercise.

Complexity of languages

Using O, we can classify languages rather than TMs.

Let $\mathsf{TIME}(f(n))$ be the set of languages L such that there exists a DTM M deciding it with $t_M(n) = O(f(n))$.

TIME is inclusive: If $f \in O(g)$, then $\mathsf{TIME}(f) \subseteq \mathsf{TIME}(g)$.

"Easy" to show that $L \in \mathsf{TIME}(f(n))$, if true; "hard" to show that $L \notin \mathsf{TIME}(g(n))$.

Computational models

So far, DTMs. What about multitape DTMs, and NTMs?

We will define interesting classes on NTMs too.

However, MDTMs and NTMs both reduce to DTMs.

The reductions, however, produce machines that have very different worst-case running times.

Simulating MDTMs, complexity

Theorem (Thm. 7.8)

Let $t(n)$ *be a function such that* $t(n) \geqslant n$. *For every MDTM with running time* $t(n)$ *there exists an equivalent DTM with running time in* $O(t(n)^2)$.

The proof is by analyzing the reduction you've already seen.

All the tapes of the MDTM are stored sequentially, with delimiters.

Estimating running time

What does our DTM do to simulate one step of the MDTM?

- Read whole tape to find symbols under heads (ReadScan)
- Scan and update whole tape (WriteScan)
- If necessary, shift entire tape to the right (Shift)

We need two things: Bound on the scans, and bound on the shifts.

Estimating running time

What does our DTM do to simulate one step of the MDTM?

- Read whole tape to find symbols under heads (ReadScan)
- Scan and update whole tape (WriteScan)
- If necessary, shift entire tape to the right (Shift)

We need two things: Bound on the scans, and bound on the shifts.

Scan cost: $t(n) \times k$, $t(n)$ for each of k tapes of the MDTM.

Shift cost: $t(n) \times k$ per shift.

Total: $t(n) \times k + t(n) \times k \times k$ for each step. k constant, $t(n)$ steps, so $O(t(n)^2)$ bound holds.

NTMs, running time

Need worst-case running time definition.

NTMs accept if some branch accepts. If we sum the running time of all branches, not so interesting.

Instead, let the running time be the max number of steps used in any branch.

Simulating NTMs, complexity

Theorem (Thm 7.11)

Let $t(n) \geqslant n$ *be a function. For ever NTM with running time* $t(n)$ *there exists an equivalent DTM with running time* $2^{O(t(n))}$.

The reduction is high-level.

We simulate every branch, record result somewhere on tape (1 bit).

Estimating NTM simulation running time

Length of branch times number of them.

Length at most $t(n)$. Number of branches?

Estimating NTM simulation running time

Length of branch times number of them.

Length at most $t(n)$. Number of branches?

Let b be the number of transitions in the *definition* of the NTM.

Then number of branches at most $b^{t(n)}$. This gives $O(t(n) \times b^{t(n)})$ running time. b is a constant.

We need $2^{O(t(n))} = 2^{ct(n)}$ for some c. $t(n), b^{t(n)} \in O(b^{t(n)})$, so we have $O(b^{2t(n)})$. Take $\log_2(b)$ into the exponent to get a power of two.

Computational models, summary

DTMs and MDTMs are *polynomial-time equivalent*.

For NTMs, we needed an exponential amount of time.

No algorithm is known to do better.

The fact that we end up with *only* an exponential amount of time is not a coincidence.

Polynomial time equivalence

We want to abstract away details of machines.

We generally consider all models that are polynomial-time equivalent to DTMs to be equivalent for complexity theory.

Since we are interested in *problems*, we also abstract away the details of input encoding.

Care must be taken with numbers: The input size of the number is not the value of the number.

Summing up

We measure worst-case time complexity of an algorithm or problem:

- as a function of input size
- disregarding constants and lower-order terms
- abstracting away encoding and using abstract TMs as model.

Using the notation $\mathsf{TIME}(f(n))$, we can classify problems.

Some deterministic classes

Logarithmic time: $\mathsf{TIME}(\log n)$. Binary search is in this class.

Linear time: $\mathsf{TIME}(n)$. Boring class.

Polynomial time (P): $\bigcup_{k \in \mathbb{N}} \mathsf{TIME}(n^k)$.

Exponential time (EXPTIME): $\bigcup_{k \in \mathbb{N}} \mathsf{TIME}(2^{n^k})$.

Some nondeterministic classes

All of the above can be done for NTMs. Let $\mathsf{NTIME}(f(n))$ be the class of languages L such that there exists an NTM M deciding L with $t_M(n) = O(f(n))$.

Nondeterministic polynomial time (NP): $\bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(n^k)$.

Nondeterministic exponential time (NEXPTIME): $\bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(2^{n^k})$.

Extra: Can we speak of "the" complexity of a language?

Recall the set $L_t = \{t_M \mid M \text{ decides } L\}$.

Tempting to look for the smallest function modulo O: $f \leqslant g$ iff $f \in O(g)$.

Extra: Can we speak of “the” complexity of a language?

Recall the set $L_t = \{t_M \mid M \text{ decides } L\}$.

Tempting to look for the smallest function modulo O: $f \leqslant g$ iff $f \in O(g)$.

Surprisingly, this does not always exist!

Blum’s speedup theorem (short version): There exists a decidable language L such that if L is decidable in $\mathsf{TIME}(t_M)$, it is also decidable in $\mathsf{TIME}(O(\log t_M))$.