COMP3630/6360: Theory of Computation
Semester 1, 2022
The Australian National University

Time Complexity

This lecture covers Chapter 10 of HMU: Time Complexity

- Determinisitic Time Complexity
- Non-deterministic Time Complexity
- The classes P and NP

Additional Reading: Chapter 10 of HMU.

Time Complexity

Exanple

We know that $A = \{ 0^i 1^i \mid i \in \mathbb{N} \}$ is a CFL and decidable, e.g. by TM M_1 which on input w:

1. Scan w and reject if anything not in $\{\sqcup, 0, 1\}$ or 10 is found.
2. Repeat as long as there are 0s and 1s on the tape:
   - Scan across and replace with blanks both the leftmost 0 and the rightmost 1.
3. If either 0s or 1s are left reject, otherwise accept.

How much time does M_1 need, as a function f of the length of the input word w?

w	ϵ	01	$0^2 1^2$	$0^3 1^3$	$0^4 1^4$	$0^5 1^5$
$f(\lvert w \rvert)$	2	8	19	34	53	76

Definition 1.1

The *time complexity* of a deterministic TM that halts on all inputs is the function $f : \mathbb{N} \longrightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n.

Example 1.2

For M_1, it seems that $f(2k) = f(2(k-1)) + 4k + 3$ for $k > 1$.

Asymptotic Notation (Big-$\mathcal{O}$)

The exact running time function is often too complicated. The highest order terms dominate the function eventually, so we can ignore the other terms.

Definition 1.3

Let $f, g : \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$ Say that $f(n) = \mathcal{O}(g(n))$ if there exist $c, n_0 > 0$ such that for all $n \geq n_0$

$$f(n) \leq c \cdot g(n) \ .$$

The function g is an (*asymptotic*) *upper bound* for f.
Bounds of the form n^c for some $c > 0$ are called *polynomial bounds*; those of the form $2^{(n^\delta)}$ are called *exponential bounds* when $\delta \in \mathbb{R}$ is positive.

Examples 1.4

- $5n^3 + 2n^2 + 22n + 6 = \mathcal{O}(n^3)$
- f from M_1 is $\mathcal{O}(n^2)$.

Big-$\mathcal{O}$ vs. $\log$

We may safely omit the base of logarithms in the big-$\mathcal{O}$ notation because

$$\log_a n = \frac{\log_b n}{\log_b a} \quad .$$

Small-o Notation

Definition 1.5

Let $f, g : \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$ Say that $f(n) = o(g(n))$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \ .$$

that is, for any $c > 0$ there exist $n_0 > 0$ such that $f(n) < c \cdot g(n)$, for all $n \geq n_0$.

Observe that

1. $f = \mathcal{O}(f)$ but $f \neq o(f)$.
2. $f = o(g) \Rightarrow f = \mathcal{O}(g)$ but in general $f = o(g) \not\Leftarrow f = \mathcal{O}(g)$

Time Complexity Classes

Definition 2.1

Let $t : \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$. Define the *time complexity class*, $\mathbf{TIME}(t(n))$ to be the collection of all languages that are decidable by an $\mathcal{O}(t(n))$ time TM.

Example 2.2

Recall $A = \left\{ 0^i 1^i \mid i \in \mathbb{N} \right\}$. Our analysis of M_1's running time showed that $A \in \mathbf{TIME}(n^2)$.

Example 2.3

Could we do better for A?
Is there a TM that decides A asymptotically more quickly, that is, is $A \in \mathbf{TIME}(t(n))$ for some $t = o(n^2)$?
Consider M_2, which on input w:

1. Scan w left to right and reject if 10 occurs as a substring. $\mathcal{O}(n)$
2. Repeat as long as both 0s and 1s are on the tape: $\mathcal{O}(\log n)$
   1. Scan from right to left and reject if there's an odd number of non-Xs on the tape. $\mathcal{O}(n)$
   2. Scan from left to right and replace every other 0 by an X, beginning with the first 0. Then do the same for the 1s. $\mathcal{O}(n)$
3. If neither 0s nor 1s are left accept, otherwise reject. $\mathcal{O}(n)$

So $L(M_2) = A \in \mathbf{TIME}(n \log n)$.

Example 2.4

Compare the running times (step numbers) of M_1 and M_2.

w	ϵ	01	$0^2 1^2$	$0^3 1^3$	$0^4 1^4$	$0^5 1^5$
$f_{M_1}(\lvert w \rvert)$	2	8	19	34	53	76
$f_{M_2}(\lvert w \rvert)$	1	15	45	63	117	141

So M_1 beats M_2 at least for short inputs. For $0^{20}1^{20}$ this is no longer the case.

Example 2.5

Could we do still better for A?
Is there a TM that decides A asymptotically more quickly, that is, is $A \in \mathbf{TIME}(t(n))$ for some $t = o(n \log n)$?
We won't prove this here, but the answer is no, not with a deterministic single-tape TM.
Consider the two-tape TM M_3, which on input w:

1. Scan from left to right and copy 0s onto the second tape until the first 1 occurs. $\mathcal{O}(n)$
2. Keep scanning left to right across the 1s while scanning right to left on the second tape. $\mathcal{O}(n)$
3. Accept if both heads encounter their first blank at the same time, otherwise reject. $\mathcal{O}(1)$

So $L(M_3) = A$ "$\in$" $\mathbf{TIME}(n)$.

Complexity vs. Computability

- In computability theory we proved that the various TM models (deterministic vs non-deterministic, single-tape vs multi-tape) were equally powerful.
- In complexity theory the choice of TM models affects the time complexity of languages.

Multi-Tape TM vs. Single-Tape TM

Theorem 2.6

Let $t : \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$ be such that $\forall n \in \mathbb{N}\,(t(n) \geq n)$. Every t-time multi-tape TM has an equivalent $\mathcal{O}((t(n))^2)$ time single-tape TM.

Proof.

By analysing the time complexity of the construction given to show that every multi-tape TM M has an equivalent single-tape TM S.

Since for every step of M S might have to scan all of the tape used so far, the single step number k of M may cost S up to $\mathcal{O}(k)$ steps. Hence the running time of S on an input of length n is $\mathcal{O}(\sum_{i=1}^{t(n)} i) = \mathcal{O}(\frac{t(n)\cdot t(n)+1}{2}) = \mathcal{O}((t(n))^2)$ $\square$

Non-Deterministic TM vs. (Deterministic) TM

The *running time* of a deciding non-deterministic TM N on an input word w is the maximum number of steps N uses on any branch of its computation tree when starting on w.

Theorem 2.7

Let $t : \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$ be such that $\forall n \in \mathbb{N}\,(t(n) \geq n)$. Every t-time non-deterministic TM has an equivalent $2^{\mathcal{O}(t)}$ time single-tape TM.

Proof.

By analysing the time complexity of the construction given to show that every non-deterministic TM N has an equivalent deterministic TM S.

- For inputs of length n the computation tree of N has depth at most $t(n)$.
- Every tree node has at most b children, where $b \in \mathbb{N}$ depends on N's transition function. Thus the tree has no more than $b^{t(n)+1}$ nodes.
- S may have to explore all of them, in a breadth first fashion. Each exploration may take $\mathcal{O}(t(n))$ steps (from the root to a node).
- So all explorations together may take $\mathcal{O}(t(n)) \cdot \mathcal{O}(b^{t(n)+1}) = 2^{\mathcal{O}(t(n))}$ time.

□

P

Further study reveals that all deterministic models of computation are time equivalent up to some polynomial. This motivates

Definition 2.8

$$\mathbf{P} = \bigcup_{k \in \mathbb{N}} \mathbf{TIME}(n^k)$$

Examples in **P**

$$PATH = \{ \langle G, s, t \rangle \mid t \text{ is reachable from } s \text{ in directed graph } G \}$$

$$RELPRIME = \{ \langle x, y \rangle \mid x, y \in \mathbb{N} \wedge \gcd(x, y) = 1 \}$$

Details are in [Sipser2006].

Examples in **P** cont.

Theorem 2.9

Every CFL is in **P**.

Proof.

For CFG G in CNF the CYK algorithm runs in $\mathcal{O}(|w|^3)$ time on w. □

Beyond (?) **P**

$$HAMPATH = \left\{ \langle G, s, t \rangle \;\middle|\; \begin{array}{l} \text{Directed graph } G \text{ has a} \\ \text{Hamiltonian path from } s \text{ to } t \end{array} \right\}$$

$$COMPOSITES = \{ \langle x \rangle \mid \exists p, q \in \mathbb{N}_{>1} \, (x = p \cdot q) \}$$

Verifying an answer is often much easier than finding it.

Definition 2.10

A *verifier* for a language A is an algorithm V, where

$$A = \{\, w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \,\}$$

We measure the running time of a verifier only in terms of the length of w, not that of the *certificate* (or *proof*) c. Language A is *polynomially verifiable* if it has a polynomial time verifier.

Examples 2.11

For *HAMPATH* a certificate for $\langle G, s, t \rangle$ could be the sequence of nodes forming a Hamiltonian path from s to t in G.

For *COMPOSITES* a certificate for $\langle x \rangle$ could be a non-trivial divisor of x.

NP

Definition 2.12

NP is the class of languages that have polynomial time verifiers.

Theorem 2.13

A language is in **NP** *iff it is decided by some non-deterministic polynomial time TM.*

Proof.

"$\subseteq$:" guess the certificate.
"$\supseteq$:" use the accepting branch of the computation tree as certificate. $\square$

NTIME

Definition 2.14

NTIME$(t(n))$ is the class of languages decided by an $\mathcal{O}(t(n))$ time non-deterministic TM.

Corollary 2.15

$$\mathbf{NP} = \bigcup_{k \in \mathbb{N}} \mathbf{NTIME}(n^k)$$

Example: Cliques

A *clique* in an undirected graph is a fully connected subgraph. A k-clique is a clique with k nodes.

$$CLIQUE = \{ \langle G, k \rangle \mid \text{Undirected graph } G \text{ contains a } k\text{-clique} \}$$

is in **NP**.

Proof.

The certificate is a (representation of a) k-clique. □