COMP3630 / COMP6363

week 7: **Time Complexity**

This Lecture Covers Chapter 10 of HMU: Time Complexity

slides created by: Dirk Pattinson, based on material by
Peter Hoefner and Rob van Glabbeck; with improvements by Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

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

Additional Reading: Chapter 10 of HMU.

Runtime of a TM (Example)

Example

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

1. Scan w and reject if anything not in $\{B, 0, 1\}$ 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	$0^2 1^2$	$0^3 1^3$	$0^4 1^4$	$0^5 1^5$
$f(\|w\|)$	2	8	19	34	53	76

The exact numbers depend on how exactly the machine is implemented. (But the idea should be clear.) Also note that we'd need the "time" (number of executed steps) for all inputs, not just the ones from L as given here as example.

Running Time of (Deterministic) TM

Definition 10.1.1

The running time 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.

Notes:

- The running time is a function taking some input – because each TM also takes an input! So the TM's runtime depends on it.
- Please note that n is the length of the input!

Example 10.1.2

For M_1, it seems that $f(2k) = f(2(k-1)) + 4k + 1$ for $k > 1$. (Though this assumes that we always start from the left, but we could do better.)

As noted earlier, we do need to know f for all possible lengths, not just for even-length strings, and not just for $k > 1$ (but also $k = 0$ and $k = 1$).

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 10.1.3

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

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

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 for Big-$\mathcal{O}$ Notation

Example 10.1.4

- $5n^3 + 2n^2 + 22n + 6 = \mathcal{O}(n^3)$
- f from M_1 (which was $f(2k) = f(2(k-1)) + 4k + 1$) is in $\mathcal{O}(n^2)$.

$\log$ in the Big-$\mathcal{O}$ Notation

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

$$\log_a n = \frac{\log_b n}{\log_b a} = \frac{1}{\log_b a} \cdot \log_b n \ .$$

(So two log functions with different bases just differ in a constant multiplier that does not depend on the input.)

Small-o Notation

Definition 10.1.5

Let $f, g : \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$. We say that $f(n) = o(g(n))$ (or $f(n) \in o(g(n))$) 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 $n \geq n_0$.

In comparison:
$f(n) = \mathcal{O}(g(n))$ if there exist $c, n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ 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 $f = o(g) \not\Leftarrow f = \mathcal{O}(g)$

Examples:

- $n \neq o(2n)$ (although $2n$ grows faster than n)
- $n = o(\frac{1}{2} n \log n)$ and $n \log n = o(n^2)$

Time Complexity Classes (Definition)

Definition 10.2.1

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

Example 10.2.2

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

Time Complexity Classes (Example)

Example 10.2.3

Could we do better for L?

Is there a TM that decides L asymptotically more quickly, that is, is $L \in \textbf{TIME}(t(n))$ for some $t = o(n^2)$?

Consider M_2, which on input w does:

1. Scan w left to right and reject if `10` occurs as a substring. $\mathcal{O}(n)$
2. Repeat as long as both `0`s and `1`s are on the tape: $\mathcal{O}(\log n)$
    1. Scan from right to left and reject if there is an odd number of non-`X`s 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 `1`s. $\mathcal{O}(n)$
3. If neither `0`s nor `1`s are left accept, otherwise reject. $\mathcal{O}(n)$

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

Example

Example 10.2.4

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

Example 10.2.5

Could we do still better for L?

Is there a TM that decides L asymptotically more quickly, that is, is $L \in \mathbf{TIME}(t(n))$ 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 does:

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 $L(M_3) = L$ "$\in$" $\mathbf{TIME}(n)$. (If we could use a multi-tape TM.)

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. (As you have just seen! Provided you believe that there's no single-tape TM running in $\mathcal{O}(n)$.)
- So it's important to differentiate between expressiveness and complexity!

Multi-Tape TM vs. Single-Tape TM

Theorem 10.2.6

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 $\mathcal{O}(\sum_{i=1}^{t(n)} i) = \mathcal{O}(\frac{t(n) \cdot (t(n)+1)}{2}) = \mathcal{O}((t(n))^2)$ □

Running Time of Non-Deterministic TM

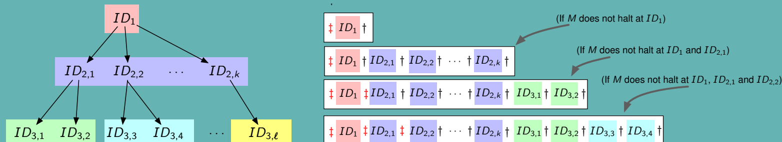
The running time of a deciding non-deterministic TM N on an input word w is the maximum number of steps N uses on any branch of its computation tree when starting on w.

Theorem 10.2.7

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 $2^{\mathcal{O}(t(n))}$ 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. (Cf. week 5)



□

Running Time of Non-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 10.2.7

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 $2^{\mathcal{O}(t(n))}$ 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. □

Our First Complexity Class: **P**

We are interested in problems that can be decided efficiently, i.e., in polynomial time.
Formally:

Definition 10.2.8

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

Note: All deterministic models of computation are time equivalent up to some polynomial! (This thus further motivates this definition.)

Examples in **P**

$$PATH = \{ \langle G, s, t \rangle \mid t \text{ is reachable from } s \text{ in directed graph } G \}$$
$$RELPRIME = \{ \langle x, y \rangle \mid x, y \in \mathbb{N} \wedge \gcd(x, y) = 1 \}$$

gcd means *greatest common divisor*. E.g.,

- $\langle 3, 5 \rangle \in RELPRIME$, since 3 and 5 are prime, so $\gcd(3, 5) = 1$
- $\langle 8, 21 \rangle \in RELPRIME$, since $8 = 2 \cdot 4$ and $21 = 3 \cdot 7$, so $\gcd(8, 21) = 1$
- $\langle 9, 12 \rangle \notin RELPRIME$, since $9 = 3 \cdot 3$ and $12 = 2 \cdot 2 \cdot 3$, so $\gcd(9, 12) = 3$

Details are in [Sipser2006].

Examples in **P** cont.

Theorem 10.2.9

Every CFL is in **P**.

First, what does that mean? When is a language in **P**?

- According to Definition 10.2.8, **P** is the set of languages for which there exists a TM that decides it in polytime.
- Thus, $L \in \mathbf{P}$ is true once there exists one representation (grammar or automaton) of it for which there exists a decision procedure (a TM) running in polytime. This is even true if there exist representations generating L for which deciding L is not in **P**!
- Thus, for a CFL L to show that $L \in \mathbf{P}$, we pick the most suitable representation!

Proof.

Let L be context-free. We thus know that there exists a CFG G in Chomsky Normal Normal (CNF). Let G be given. Now run CYK on the input w, taking $\mathcal{O}(|w|^3)$ time. □

Note that this proof would even be correct if converting a CFG G into CNF would take exponential time! (Due to the arguments above.) But it's not! Conversion is polytime.

Beyond (?) **P**

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

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

A Hamiltonian path is a path in a graph that visits each vertex exactly once.

Verifying an answer is often much easier than finding it.

Definition 10.2.10

A verifier for a language A is an algorithm V, such that

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

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

Note that for many problems certificates are 'directly' those properties that are being demanded (e.g., a Hamiltonian path, the divisor, etc.), but it can also be another property, from which membership follows logically. (See week 12!)

Examples 10.2.11

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 10.2.12

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

Theorem 10.2.13

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

Proof.

$\rightarrow$: Let $L \in NP$. We thus know that a (deterministic) verifier V exists, and for each word $w \in L$ a certificate that's poly-length in w. Design a poly-time NTM M with $L = L(M)$ as follows. First, M non-deterministically generates all possible certificates (i.e., each run produces one certificate). Then, we switch into the second phase which implements V, which in turn verifies the written certificate and accepts or rejects accordingly.
$\leftarrow$: Let M be a poly-time NTM with $L = L(M)$. Then there must be a sequence of configurations, bounded by a polynomial in the length of each input, which ends in an accepting state/configuration. Use this sequence as certificate: Design a deterministic verifier V, which tests whether a given sequence of configurations is valid with regard to M and ends in an accepting configuration. Accept/reject accordingly. □

Note that one can also switch what is a definition and what is the theorem!

NTIME

Definition 10.2.14

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

Corollary 10.2.15

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

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. □