3P17/4P17 COMPUTABILITY 2014-2015 Lecture Handout 11

Time complexity

Let M be a deterministic TM that halts on all inputs, i.e., is a decider.

Recall $t_M : \Sigma^* \to \mathbb{N}$ is a function where $t_M(\sigma)$ is the number of steps M takes to halt when given input σ.

The **worst-case time complexity** of M is the function $f : \mathbb{N} \to \mathbb{N}$ where

$$f(n) = \max\{t_M(\sigma) : \sigma \in \Sigma^*, |\sigma| = n\}.$$

Note, this is often shortened to **time complexity** or just **complexity**. It is often also referred to as the **running time**.

Whichever of these it is called, it always answers the same question: Over all inputs of length n, how long does the longest computation take?

Note: there are other time complexity questions, such as the average time complexity, or best case time complexity.

Obviously best-case complexity $\leq$ average case complexity $\leq$ worst-case complexity.

We shall be dealing mainly with worst case.

Let $t : \mathbb{N} \to \mathbb{R}^+$. Define the **time complexity class TIME($t(n)$)** to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Examples

The following TM will decide the language $L = \{\mathtt{0}^{\mathtt{k}}\mathtt{1}^{\mathtt{k}} : \mathtt{k} \geq \mathtt{0}\}$ over alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$:

M_1
The tape alphabet is $\Gamma = \{\mathtt{0}, \mathtt{1}, *, \triangleright, \sqcup\}$. On input σ:

(0) Scan through tape checking if there is a `0` to the right of a `1`. If so, reject, if not, return head to location 1, leaving the input tape unchanged.

(1) Scan through tape from location 1 doing the following:

- If a `0` is encountered, replace it with a $*$ go to (2)
- If a `1` is encountered, reject
- If there are no `0`'s and no `1`'s encountered, then accept.

(2) Continue scanning looking for a `1`.

- If a `1` is encountered replace it with a $*$, move head to location 1 and go to (1).
- If no `1` is found, then reject.

Theorem 1. *The time complexity of M_1 is $O(n^2)$.*

Proof. (0) takes $O(n)$ time because it only needs to make a single pass through the input, remembering if it has seen a `1` so that it can reject if it sees a `0` thereafter.

It is executed once. Therefore, it contributes a total $O(n)$ to the time complexity.

Each iteration of (1) and (2) takes $O(n)$ time, because together they make a single pass through the tape content, or terminate before reaching the end. Each non-blank square only gets seen at most once in each iteration, and the tape contents is not changed.

Each of (1) and (2) is executed at most $n+1$ times. Therefore, they contribute a total $O(n^2)$ time complexity.

The total complexity of this algorithm is therefore $O(n^2) + O(n) = O(n^2)$. □

Corollary 2. $L \in \boldsymbol{TIME}(n^2)$.

In fact, there is a matching lower bound:

Theorem 3. *The time complexity of* M_1 *is* $\Omega(n^2)$

Proof. If the input is of the form $\mathtt{0}^{n/2}\mathtt{1}^{n/2}$ for even n, then there will by an iteration of (1)-(2) for each $\mathtt{0}$, and in each iteration, it will have to scan through at least $n/2$ locations. Therefore, it will take time at least $n^2/4 = \Omega(n^2)$.

Hence, in the worst case over all σ with $|\sigma| = n$, $t_{M_1}(\sigma) = \Omega(n)$.

□

Corollary 4. M_1 *has time complexity* $\Theta(n^2)$.

Can we do better? Yes, the following algorithm has complexity $O(n \log n)$:

$\underline{M_2}$
The tape alphabet is $\Gamma = \{\mathtt{0}, \mathtt{1}, *, \triangleright, \sqcup\}$. On input σ:

(0) Scan through tape checking if there is a $\mathtt{0}$ to the right of a $\mathtt{1}$. If so, reject, if not, return head to location 1, leaving the input tape unchanged.

(1) From location 1, scan through the tape checking if the total number of $\mathtt{0}$s and $\mathtt{1}$s is zero or odd.

If it is zero, accept.

If it is odd, reject

(2) From location 1, scan through the tape replacing with $*$ every other $\mathtt{0}$ starting with the first $\mathtt{0}$.

(3) From location 1, scan through the tape replacing with $*$ every other $\mathtt{1}$ starting with the first $\mathtt{1}$.

(4) Go to (1).

Firstly, let's check that M_2 does, in fact, give the right answer:

Theorem 5. M_2 *decides* L.

Proof. If $\sigma = \mathtt{0}^k\mathtt{1}^k$ for some $k \geq 0$, then it will pass (0).

Then if $k = 0$, it will, rightly, be accepted at (1). If $k > 0$, then their total will be $2k$ so will it will pass to (2) which replaces $\lceil k/2 \rceil$ $\mathtt{0}$s, passing to (3) which crosses off the same number of $\mathtt{1}$s. It then passes to (4) which takes it back to (1) and the cycle repeats, always removing at least half the remaining number of $\mathtt{0}$s and $\mathtt{1}$s, leaving the same number of $\mathtt{0}$s and $\mathtt{1}$s, after (3). Eventually, (1) counts zero $\mathtt{0}$s and $\mathtt{1}$s and accepts.

So if $\sigma \in L$ M_2 accepts it.

If $\sigma \notin L$, then either it gets rejected at (0) or it is of the form $\sigma = \mathtt{0}^r\mathtt{1}^s$ where $r > s$, or of the form $\sigma = \mathtt{0}^r\mathtt{1}^s$ where $r < s$.

Suppose it is the former $r > s$. Then we get a sequence $(r(0), s(0)), (r(1), s(1)), \ldots$ where $(r(i), s(i))$ is the number of `0`s and `1`s just before (1) has been entered for the $i+1$'th time. So $(r(0), s(0)) = (r, s)$.

If they sum to an even number then they are of the form $(2a+1, 2b+1)$ or $(2a, 2b)$ with a, b integers and $a > b$. Then the next terms in the sequence will be (a, b). Since $r(i)$ and $s(i)$ are non-negative integers, strictly decreasing in i and $r(i) > s(i)$ always, it must be the case that eventually $r(i) + s(i)$ is odd. Then M_2 will reject.

For $r < s$ we reason similarly.

Thus, M_2 decides L.

□

Now we analyse the time complexity:

Theorem 6. *The time complexity of M_2 is $O(n \log n)$.*

Proof. Again, (0) contributes a total $O(n)$.

Each execution of (1)-(3) is $O(n)$ because the TM always moves right, and so sees each non-blank square at most once. (4) is $O(1)$.

Every next iteration cuts the number of `0`s and `1`s by half.

Thus, there are at most $O(\log_2 n)$ iterations.

Hence, the total time is $O(n \log_2 n) + O(n) = O(n \log n)$. □

Observe we dropped the base-2 in the logarithm on the RHS because changing base incurs only a constant factor, which is suppressed by O notation.

Corollary 7. $L \in \boldsymbol{TIME}(n \log n)$.

Comparison between Turing machine models

The definition of time complexity for multitape Turing machines is much like for single-tape ones.

Theorem 8. *Let $f(n)$ be a function where $f(n) \geq n$. Then every $f(n)$ time multitape decider has an equivalent $O(f^2(n))$ time single tape decider.*

We shall say a non-deterministic Turing machine is a **decider** if it always halts. (Note, this is different to Sipser's definition, but it's not important).

For a NDTM N deciding a language L, we shall measure the time complexity of N on σ as $t_N(\sigma) = n$ if on input σ, N generates a sequence of configurations $\mathcal{S}_0, \mathcal{S}_1, \ldots$ that terminates with $\mathcal{S}_n$.

We can give a equivalent definition based on the computation tree generated by a NDTM: $t_N(\sigma) = n$ if N accepts σ and the shortest accepting path from the root is length n, or N rejects σ and the longest rejecting path from the root is n (recall in this case all paths from the root lead to reject).

Note, authors tend to have different definitions of time complexity for a NDTM, but the differences are not very important.

With $t_N(\sigma) = n$ defined, the complexity is much the same as for single-tape deterministic Turing machines. That is, for a deciding NDTM N, the (worst-case) time complexity is $f(n) = \max\{t_N(\sigma) : \sigma \in \Sigma^*, |\sigma| = n\}$.

Theorem 9. *Let $f(n)$ be a function where $f(n) \geq n$. Then every $f(n)$ time non-deterministic single-tape decider has an equivalent $2^{O(f(n))}$ single tape decider.*