Time Complexity Classes

Measuring Complexity

Complexity of a problem $=$ **the efficiency of the best algorithm for the problem**

Measure the efficiency by **time** or **space**, or both

Analyze the efficiency by the **growth of the function that relates the amount of resources used to the input size**

Worst-case analysis $\cdots$ analyze the function that maps each nonnegative integer n to the *maximum amount of resources used for solving any input of size* n with the best algorithm known

An alternative is **average case analysis**

An example: $L = \{0^n 1^n \mid n \geq 0\}$

Algorithm A, for a **1-tape TM**

Match and erase the outermost non-blank symbols

Notation: C : the symbol currently scanned

$[H \Rightarrow]$ ($[H \Leftarrow]$) : moving the head to the right (left) by one cell

The Main Loop:

1. **if** $C = \sqcup$ **then accept** ; **if** $C = 1$ **then reject**
2. $C \leftarrow \sqcup$
3. **repeat** $[H \Rightarrow]$ **until** $C = \sqcup$;
4. $[H \Leftarrow]$; **if** $C \neq 1$ **then reject** ; $C \leftarrow \sqcup$
5. **repeat** $[H \Leftarrow]$ **until** $C = \sqcup$; $[H \Rightarrow]$

Big-O, small-o

Definition. *Let $f, g : \mathcal{N} \to \mathcal{R}^+$ be two functions.*

- *$f = O(g)$ if there exists a constant $c > 0$ such that for all but finitely many n, $f(n) \le cg(n)$.*
- *$f = o(g)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.*

Example: $\log n = o(n)$, $\log n \log n = o(n)$, $10n + 8 \log n = O(n)$, etc.

Note: $f = o(g)$ implies $f = O(g)$, but the converse does not necessarily hold

The running time of Algorithm A is $O(n^2)$

Can do in $O(n)$ steps if two tapes are available

Algorithm B, for a 2-tape TM

Copy the first run of 0s onto Tape 2 and measure the length of the following run of 1s

Notation: C_i, $[H_i \Rightarrow]$, $[H_i \Leftarrow]$, $i = 1, 2$

1. **(the initial check)**

   if $C_1 = \sqcup$ **then accept ; if** $C_1 = 1$ **then reject**

2. **put the left marker**

   $C_2 \leftarrow \#; [H_2 \Rightarrow]$

3. **(copying the first block)**

   while $C_1 = 0$ **do**

   - $C_2 \leftarrow 0; [H_1 \Rightarrow]; [H_2 \Rightarrow]$

4. **if** $C_1 = \sqcup$ **then reject** ;
5. **(comparing the length)**

   while $C_1 = 1$ **do**

   - **if** $C_2 = \#$ **then reject** ; $[H_1 \Rightarrow]$; $[H_2 \Leftarrow]$;
6. **if** $C_1 = \sqcup$ and $C_2 = \#$ **then accept** ; **else reject**

Algorithm B runs in $n + 1 = O(n)$ steps.

Deterministic Time Complexity Classes

Definition. *Let $t : \mathcal{N} \rightarrow \mathcal{N}$ be a function. A Turing machine M is $t(n)$ time (or $t(n)$ time-bounded) if for every $n \in \mathcal{N}$, and for every input x of length n, M on x halts within $t(n)$ steps.*

Definition. *Let $t : \mathcal{N} \rightarrow \mathcal{N}$ be a function. Define $\mathrm{TIME}(t(n)) = \{L \mid L$ is decided by an $O(t(n))$ time multi-tape Turing machine $\}$.*

$L = \{0^n 1^n \mid n \geq 1\}$ is in $\mathrm{TIME}(n)$, the **linear time** complexity class

Nondeterministic Time Complexity Classes

Definition. *Let $t : \mathcal{N} \to \mathcal{N}$ be a function. A nondeterministic Turing machine N is $t(n)$ time if for every $n \in \mathcal{N}$, and for every input x of length n, N on x halts within $t(n)$ steps along all computation paths.*

Definition. *Let $t : \mathcal{N} \to \mathcal{N}$ be a function. Define $\mathrm{NTIME}(t(n)) = \{L \mid L$ is decided by an $O(t(n))$ time nondeterministic multi-tape Turing machine $\}$.*

Relationships Among Models

Theorem. *For every $t(n) \geq n$, each $t(n)$ time multi-tape Turing machine has an equivalent $t(n)^2$ time single-tape Turing machine.*

Theorem. *For every $t(n) \geq n$, each $t(n)$ time nondeterministic Turing machine has an equivalent $2^{O(t(n))}$ time single-tape Turing machine.*