Teoria da Computação
2018/2

Álvaro Moreira
afmoreira@inf.ufrgs.br

Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brasil
http://www.inf.ufrgs.br

Contents

Contents

Measuring Complexity

If a problem is decidable, in principle it can solvable by a computer

But if a decidable problem is such that its solution requires an unreasonably large amount of **time** or **space**, it may not be solvable in practice by a computer

We will see basic elements of **(Computational) Complexity Theory** focusing on the resources time and space

Measuring Complexity

We will consider here **worst-case analysis**, i.e we consider the longest running time of all inputs of a particular length.

Definition: *Let* M *be a deterministic TM that halts on all inputs. The* ***running time*** *or* ***time complexity*** *of* M *is the function* $f : \mathbb{N} \to \mathbb{N}$*, onde* $f(n)$ *is the maximum number of steps that* M *uses on any input of length* n*. If* $f(n)$ *is the running time of* M *we say that* M *runs in time* $f(n)$ *or that* M *is an* $f(n)$ *time TM.*

In **average-case analysis**, the average of all running time of inputs of a particular length is considered.

Big-O notation

Asymptotic analysis gives the running time of an algorithm on large inputs

For instance, suppose that a more detailed running time analysis has concluded that the exact running time of an algorithm is given by the following function:

$$f(n) = 6n^3 + 2n^2 + 20n - 45$$

On very large n the term n^3 dominates $f(n)$. We say that f is asymptotically at most n^3, or using the big-O notation, that $f(n) = O(n^3)$

Definition: *Let* $f, g : \mathbb{N} \to \mathbb{R}^+$. *We say that* $\mathbf{f(n) = O(g(n))}$ *if there are positive integers* c *and* n_0 *such that, for every* $n \geq n_0$

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

When $f(n) = O(g(n))$ *we say that* $g(n)$ *is an* ***asymptotic upper bound*** *for* $f(n)$

Example

Let $f_1(n) = 5n^3 + 2n^2 + 22n + 6$. Then $f_1(n) = O(n^3)$.

And we can check that this is in agreement with the definition considering $c = 6$ and $n_0 = 10$, since it is true that, for every integer $n \geq 10$, we have that

$$5n^3 + 2n^2 + 22n + 6 \leq 6n^3$$

Note that we can also say that $f_1(n) = O(n^4)$ because n^4 is larger than n^3 so it is also an asymptotic upper bound for $f(n)$

But $f_1(n)$ is not $O(n^2)$ since there are no positive integers c and n_0 such that $f_1(n) \leq c.n^2$ for every $n \geq n_0$

Polynomial and Exponential bounds

Bounds of the form n^c for some constant $c > 0$ are called **polynomial bounds**

Bounds of the form 2^{n^δ} for some real number $\delta > 0$ are called **exponential bounds**

If a TM has a polynomial bound we say that it is a polynomial time (bounded) TM, or that it runs in (no more than) polynomial time

If a TM has an exponential bound we say that it is an exponential time (bounded) TM, or that it runs in (no more than) exponential time

Analysing algorithms

Definition: *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* $O(t(n))$ *TM*

Consider the language $A = \{0^k 1^k \mid k \geq 0\}$. Since there is a $O(n^2)$ TM than can decide a we have that $A \in \mathbf{TIME}(n^2)$

But it is also possible to construct a $O(n \log n)$ TM to decide A, hence it is also true that $A \in \mathbf{TIME}(n \log n)$ and this result cannot be improved on a single tape TM

With a two tape TM we can be even faster and decide A in $O(n)$ so $A \in \mathbf{TIME}(n)$

So the complexity of A depends on the the model of computation used !

(see chapter 7 of Sisper for the analysis of the TMs that decide A)

Models in Complexity theory

In Computability Theory, we know that (variations of) TM and many other computational models (like Lambda calculus, for instance) are all equivalent in the sense that they all decide the same class of languages

In Complexity Theory the choice of model affects the time complexity of the language

Complexity Theory classifies languages according to their time complexity. But which model we choose to measure time?

Fortunately, time requirements don't differ too much for deterministic models

And since the classification system of Complexity Theory is not too sensitive to small differences in complexity, the choice of deterministic model is not crucial

Complexity relationship among models

Theorem: Let $t(n)$ be a function, where $t(n) \geq n$. then every $t(n)$ time multitape TM has an equivalent $O(t(n)^2)$ time single tape TM.

This theorem says that the difference between the time complexity of problems measured on deterministic single tape and multitape TMs is at most a square difference, or a *polynomial* difference

Definition: *let N be a non-deterministic decider TM. The running time of N is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on an input of length n.*

Theorem: Let $t(n)$ be a function where $t(n) \geq n$. then every $t(n)$ time nondeterministic single-tape TM has an equivalent $2^{O(t(n))}$ time deterministic single-tape TM.

The theorem above says that there is at most an exponential time difference between the time complexity of problems on deterministic and nondeterministic TMs

The Class P

We now start studying the classification System of Complexity Theory with the class P

Definition: *P is the class of languages that are decidable in polynomial time on a deterministic single-tape TM. In other words,*

$$P = \bigcup_k \text{TIME}(n^k)$$

P is invariant for all models that are polynomially equivalent to the deterministic single-tape TM

P roughly corresponds to the class of problems that are realistically solvable on a computer (Cobham-Edmonds Thesis - 1964)

Of course that a solution with running time of n^{1000} is unlikely to be of practical use. Even so, considering polynomial time the limit of practical solvability has proven to be useful

Examples of Problems

See Sipser's book for examples of proofs that some problems are polynomial

The description of examples will be given in high level without details of TMs with numbered stages (for P and for all other classes from now on)

We have to give a polynomial upper bound on the number of stages that the algorithm uses when it runs on an input of size n

We also have to check that each stage executes in polynomial time on a deterministic model

Since the algorithm is described in a polynomial number of stages, each one of them executing in polynomial time, then we can conclude that it runs in polynomial time (the composition of polynomial is polynomial)

Codification

The encoding of one or more objects of the input is given using brackets $\langle , \rangle$. No particular encoding method is specified but encoding and decoding should be done in polynomial time

A graph $G = (V, E)$, for instance, can be represented a list with $|E|$ pairs of nodes, where each node is represented by a number. A pair (a, b) in the list means that there is an edge connecting nodes a and b

Another way to encode G is ussing and adjacency matrix, where the (i, j) entry with 1 means there is an edge connecting nodes i and j, and if it is 0 it means there is no edge connecting them

In a reasonable graph representation the size of the representation is polynomial in the number of nodes

Thus, if we analyse a graph G algorithm and conclude that it runs in polynomial/exponential) time in the number of nodes we know that it runs in polynomial/exponential) time in the size of the representation of G it receives as input