CS:4330 Theory of Computation

Spring 2018

Computability Theory

Time Complexity

Haniel Barbosa

The University of Iowa

Readings for this lecture

Chapter 7 of [Sipser 1996], 3rd edition. Section 7.1.

Time Complexity

- For a problem to be solvable in principle does not mean to be solvable in practice

- We will now investigate the time, memory, or other resources required for solving computational problems

- We start with the basics of time complexity theory
  - How to measure the time necessary to solve a problem
  - How to classify problems according to the amount of time required

Example computation

Consider the decidable language $A = \{0^n 1^n \mid n \geq 0\}$ and the following TM M_1 deciding A:

$M_1 =$"On input string w:

1. Scan across the tape and *reject* if a 0 appears after a 1
2. Repeat as long as both 0s and 1s remain on the tape:
    3. Scan across the tape, crossing off a single 0 and a single 1
4. If 0s still remain after all 1s have been crossed off, or if 1s still remain after all 0s have been crossed off, *reject*. Otherwise *accept*."

How much time does M_1 take to decide A?

Example computation

Consider the decidable language $A = \{0^n 1^n \mid n \geq 0\}$ and the following TM M_1 deciding A:

$M_1 =$"On input string w:

1. Scan across the tape and *reject* if a 0 appears after a 1
2. Repeat as long as both 0s and 1s remain on the tape:
    3. Scan across the tape, crossing off a single 0 and a single 1
4. If 0s still remain after all 1s have been crossed off, or if 1s still remain after all 0s have been crossed off, *reject*. Otherwise *accept*."

How much time does M_1 take to decide A?

We will measure time as a function of the number of steps TM has moved, which depends on several parameters.

Simplifying conventions

- ▷ The running time of an algorithm is a function of the length of the string representing the input on the algorithm
- ▷ In *worst-case analysis* we consider the longest running time of all inputs of a particular length
- ▷ In *average-case analysis* we consider the average of all the running times of inputs of a particular length

Time complexity of a TM

Definition

Let M be a deterministic TM that halts on all inputs. The *running time* or *time complexity of M* is a function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n)$ 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 $f(n)$ and that M is an $f(n)$ time Turing machine
- Customarily n represents the length of the input

Big-O and small-O notation

- ▷ Since the exact running time of an algorithm is often a complex expression, we usually just estimate it
- ▷ One convenient form of estimation is the so called *asymptotic analysis* which determines the running time of the algorithm on large inputs
- ▷ Consider only the highest order term of the expression, disregarding both the coefficient of that term and any lower order terms
- ▷ This is valid because the value of the highest order term dominates the value of the other terms on large inputs

Consider the function $f(n) = 6n^3 + 2n^2 + 10n + 100$.

- ▶ Disregarding the coefficient 6, we say that f is asymptotically at most n^3
- ▶ The asymptotic notation, or big-O notation, for describing this estimation is $f(n) = \mathcal{O}(n^3)$

Formally

Definition

Let f and g be functions such that $f, g : \mathbb{N} \to \mathbb{R}^+$, We say that $f(n) = \mathscr{O}(g(n))$ if positive integers c and n_0 exist such that

$$\forall n. n \geq n_0 \to f(n) \leq cg(n)$$

When $f(n) = \mathscr{O}(g(n))$ we say that $g(n)$ is an *upper bound* for $f(n)$; more precisely, $g(n)$ is an *asymptotic upper bound* for $f(n)$, to emphasize the suppression of constant factors.

Intuitively:

- $f(n) = \mathscr{O}(g(n))$ means that f is less than or equal to g if we disregard differences up to a constant factor.
- One may think of $\mathscr{O}$ as representing a suppressed constant
- In practice most functions f encountered in algorithm analysis have an obvious highest order term $h(n)$. In that case, $f(n) = \mathscr{O}(h(n))$

Interaction with logarithms

- When we use logarithms the base does not need to be specified
- Since $\log_b n = \log_2 n / \log_2 b$, we can always change the base by changing the value by a constant factor
- And we can ignore constant factors)

If

$$f_2(n) = 3n\log_2(n) + 5n\log_2(\log_2 n) + 2$$

we have $f_2(n) = \mathcal{O}(n \log n)$, since $\log n$ dominates $\log \log s$

Expressions of big-O

- Consider $f(n) = \mathscr{O}(n^2) + \mathscr{O}(n)$, where each occurrence of $\mathscr{O}$ represents a different suppressed constant. Because $\mathscr{O}(n^2)$ dominates $\mathscr{O}(n)$, $f(n) = \mathscr{O}(n^2)$
- When $\mathscr{O}$ occurs in the exponent, as in $f(n) = 2^{\mathscr{O}(n)}$, the same principle of suppressing constants by using the big-O notation applies, i.e. 2^{cn} is the upper bound of $f(n)$, for some constant c
- For expressions of the form $f(n) = 2^{\mathscr{O}(\log n)}$, using the identity $n = 2^{\log_2 n}$, from which we can deduce $n^c = 2^{c\log_2 n}$, we can see that $2^{\mathscr{O}(\log n)}$ represents an upper bound n^c for $f(n)$, for some constant c.
- Because $\mathscr{O}(1)$ represents a value that is never more than a constant, $n^{\mathscr{O}(1)}$ represents as well an upper bound n^c, for some constant c.

Polynomial and Exponential Bounds

- Bound of the form n^c for $c > 0$ are called polynomial
- Bounds of the form 2^{n^δ}, for $\delta > 0$, are called exponential bounds

Small-O notation

- ▷ Big-O notation says that a function is asymptotically *no more than* another function
- ▷ Small-O notation says that a function is asymptotically *less than* another function

Definition

For $f, g : \mathbb{N} \to \mathbb{R}^+$, we say that $f(n) = o(g(n))$ if

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

That is, $f(n) = o(g(n))$ means that for *any* real number $c > 0$ there exists n_0 such that $f(n) < cg(n)$ for all $n \geq n_0$

Intuitively $f(n) = o(g(n))$ means that $g(n)$ grows much faster than $f(n)$

Examples

Check that

- $\sqrt{n} = o(n)$
- $n = o(n \log(\log n))$
- $n \log(\log(n)) = o(n \log n)$
- $n \log n = o(n^2)$
- $n^2 = o(n^3)$

Moreover, $f(n)$ is never $o(f(n))$

Analyzing Algorithms

Consider again the TM M_1 that decides the language $A = \{0^n 1^n \mid n \geq 0\}$:
$M_1 =$"On input string w:

1. Scan across the tape and *reject* if a 0 appears after a 1
2. Repeat as long as both 0s and 1s remain on the tape:
    3. Scan across the tape, crossing off a single 0 and a single 1
4. If 0s or 1s still remain, *reject*. Otherwise *accept*."

Analyze each of the four stages separately:

- ▷ In stage 1 the machine scans the tape in n steps to verify that the inputs is of the form 0^*1^*; repositioning the tape at the left end uses another n steps. So, the total number of steps is $2n$, i.e. takes time $\mathcal{O}(n)$

Analyzing Algorithms

Consider again the TM M_1 that decides the language $A = \{0^n 1^n \mid n \geq 0\}$:
$M_1 =$"On input string w:

1. Scan across the tape and *reject* if a 0 appears after a 1
2. Repeat as long as both 0s and 1s remain on the tape:
    3. Scan across the tape, crossing off a single 0 and a single 1
4. If 0s or 1s still remain, *reject*. Otherwise *accept*."

Analyze each of the four stages separately:

- ▷ In stage 1 the machine scans the tape in n steps to verify that the inputs is of the form 0^*1^*; repositioning the tape at the left end uses another n steps. So, the total number of steps is $2n$, i.e. takes time $\mathcal{O}(n)$
- ▷ Each scan of the tape in stages 2 and 3 is performed in $\mathcal{O}(n)$. Because each scan crosses off two symbols, a 0 and a 1, at most $n/2$ scans occur. Therefore the total time taken is $(n/2)\mathcal{O}(n) = \mathcal{O}(n^2)$

Analyzing Algorithms

Consider again the TM M_1 that decides the language $A = \{0^n 1^n \mid n \geq 0\}$:
$M_1 =$"On input string w:

1. Scan across the tape and *reject* if a 0 appears after a 1
2. Repeat as long as both 0s and 1s remain on the tape:
    3. Scan across the tape, crossing off a single 0 and a single 1
4. If 0s or 1s still remain, *reject*. Otherwise *accept*."

Analyze each of the four stages separately:

- ▷ In stage 1 the machine scans the tape in n steps to verify that the inputs is of the form 0^*1^*; repositioning the tape at the left end uses another n steps. So, the total number of steps is $2n$, i.e. takes time $\mathcal{O}(n)$
- ▷ Each scan of the tape in stages 2 and 3 is performed in $\mathcal{O}(n)$. Because each scan crosses off two symbols, a 0 and a 1, at most $n/2$ scans occur. Therefore the total time taken is $(n/2)\mathcal{O}(n) = \mathcal{O}(n^2)$
- ▷ In stage 4 the machine makes a single scan to decide whether to accept or reject taking $\mathcal{O}(n)$ time

Analyzing Algorithms

Consider again the TM M_1 that decides the language $A = \{0^n 1^n \mid n \geq 0\}$:
$M_1 =$"On input string w:

1. Scan across the tape and *reject* if a 0 appears after a 1
2. Repeat as long as both 0s and 1s remain on the tape:
    3. Scan across the tape, crossing off a single 0 and a single 1
4. If 0s or 1s still remain, *reject*. Otherwise *accept*."

Analyze each of the four stages separately:

- ▷ In stage 1 the machine scans the tape in n steps to verify that the inputs is of the form 0^*1^*; repositioning the tape at the left end uses another n steps. So, the total number of steps is $2n$, i.e. takes time $\mathcal{O}(n)$
- ▷ Each scan of the tape in stages 2 and 3 is performed in $\mathcal{O}(n)$. Because each scan crosses off two symbols, a 0 and a 1, at most $n/2$ scans occur. Therefore the total time taken is $(n/2)\mathcal{O}(n) = \mathcal{O}(n^2)$
- ▷ In stage 4 the machine makes a single scan to decide whether to accept or reject taking $\mathcal{O}(n)$ time

The total time of M_1 on an input of length n is $\mathcal{O}(n) + \mathcal{O}(n^2) + \mathcal{O}(n) = \mathcal{O}(n^2)$

Notations for classifying languages

Definition

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. The time complexity class $\mathrm{TIME}(t(n))$ is the collection of languages that are decidable by an $\mathcal{O}(t(n))$ time Turing machine.

- Since language $A = \{0^n 1^n \mid n \geq 0\}$ is decided by M_1 in $\mathcal{O}(n^2)$ time, $A \in \mathrm{TIME}(n^2)$
- $\mathrm{TIME}(n^2)$ contains all languages decidable in $\mathcal{O}(n^2)$ time

Notations for classifying languages

Definition

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. The time complexity class $\mathrm{TIME}(t(n))$ is the collection of languages that are decidable by an $\mathcal{O}(t(n))$ time Turing machine.

- ▷ Since language $A = \{0^n 1^n \mid n \geq 0\}$ is decided by M_1 in $\mathcal{O}(n^2)$ time, $A \in \mathrm{TIME}(n^2)$
- ▷ $\mathrm{TIME}(n^2)$ contains all languages decidable in $\mathcal{O}(n^2)$ time
- ▷ Is there a TM that decides A asymptotically faster? I.e. is A in $\mathrm{TIME}(t(n))$ for $t(n) = o(n^2)$?
- ▷ One could cross 2 0s and 2 1s in stage 3, which cuts the number of scans by half. However this a constant factor, which does not affect the asymptotic running time.

Improving asymptotic complexity

Let M_2 be the following TM:
M_2 =“On input string w:

1. Scan across the tape and *reject* if a 0 appears after a 1
2. Repeat as long as some 0s and some 1s remain on the tape:
    3. Scan across the tape, checking whether the total number of 0s and 1s remaining on the tape is even or odd. If it is odd, *reject*
    4. Scan again across the tape, crossing every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1
5. If no 0s and no 1s remain on the tape, *accept*, otherwise *reject*.”

- ▷ Stages 1 and 5 take $\mathcal{O}(n)$ time
- ▷ Stage 4 crosses off at least half the 0s and 1s each time it is executed, so at most $1+\log_2 n$ iterations of the repeat loop occur before all get crossed off
- ▷ Therefore the total time of stages 2, 3, and 4 is $(1+\log_2 n)\mathcal{O}(n) = \mathcal{O}(n \log n)$

Improving asymptotic complexity

Let M_2 be the following TM:
$M_2 =$"On input string w:

1. Scan across the tape and *reject* if a 0 appears after a 1
2. Repeat as long as some 0s and some 1s remain on the tape:
    3. Scan across the tape, checking whether the total number of 0s and 1s remaining on the tape is even or odd. If it is odd, *reject*
    4. Scan again across the tape, crossing every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1
5. If no 0s and no 1s remain on the tape, *accept*, otherwise *reject*."

- ▷ Stages 1 and 5 take $\mathcal{O}(n)$ time
- ▷ Stage 4 crosses off at least half the 0s and 1s each time it is executed, so at most $1+\log_2 n$ iterations of the repeat loop occur before all get crossed off
- ▷ Therefore the total time of stages 2, 3, and 4 is $(1+\log_2 n)\mathcal{O}(n) = \mathcal{O}(n\log n)$

The total time of M_2 is $\mathcal{O}(n)+\mathcal{O}(n\log n) = \mathcal{O}(n\log n)$

Further improving asymptotic complexity

The following two-tape TM M_3 decides A in linear time, i.e. in $\mathcal{O}(n)$ time:
$M_3 =$"On input string w on tape 1:

1. Scan across tape 1 and *reject* if a 0 appears after a 1
2. Scan across the 0s on tape 1 until the first 1, while also copying the 0s on tape 2.
3. Scan across the 1s on tape 1 until the end o the tape is discovered. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all 1 are read, *reject*
5. If all 0s have been crossed off, *accept*, otherwise *reject*."

Further improving asymptotic complexity

The following two-tape TM M_3 decides A in linear time, i.e. in $\mathcal{O}(n)$ time:
$M_3 =$"On input string w on tape 1:

1. Scan across tape 1 and *reject* if a 0 appears after a 1
2. Scan across the 0s on tape 1 until the first 1, while also copying the 0s on tape 2.
3. Scan across the 1s on tape 1 until the end o the tape is discovered. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all 1 are read, *reject*
5. If all 0s have been crossed off, *accept*, otherwise *reject*."

- ▷ M_1 using one tape decides A in $\mathcal{O}(n^2)$
- ▷ M_2 using one tape decides A in $\mathcal{O}(n \log n)$
- ▷ M_3 using two tape decides A in $\mathcal{O}(n)$

Time complexity of A on one-tape TM is $\mathcal{O}(n \log n)$; time complexity of A on two-tape TM is $\mathcal{O}(n)$

Observations

There is an important difference between complexity theory and computability theory:

- ▷ The Church-Turing thesis implies that all reasonable models of computation are equivalent, i.e. they all decide the same class of languages

- ▷ In complexity theory the choice of model affects the time complexity of the language decided by that model (i.e. languages decided by model C in linear time are not necessarily decidable in linear time by the equivalent model C').

Complexity relations among models

Complexity relations among models show how choice of computational model can affect the time complexity of languages. We discuss this considering three computation models:

- ▷ Single-tape Turing machine
- ▷ Multitape Turing machine
- ▷ Nondeterministic Turing Machine

Multitape Turing machines

Theorem

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $\mathcal{O}(t^2(n))$ time single-tape Turing machine.

Proof idea

We know how to convert a multitape TM M into a single-tape TM S that simulates M. We show that each step of M can be simulated by S in time $\mathcal{O}(t(n))$. Since there are $\mathcal{O}(t(n))$ steps performed by M there will be $\mathcal{O}(t^2(n))$ steps performed by S.

Nondeterministic Turing Machines

Definition (Running time for Nondeterministic TMs)

Let N be a nondeterministic Turing machine decider. 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 makes on any branch of its computation on any input of length n.

Theorem

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{\mathcal{O}(t(n))}$ time deterministic single-tape Turing machine.

Proof idea

Let N be a nondeterministic TM running in time $t(n)$. We construct a deterministic TM D that simulates N by searching N's nondeterministic computation tree.

Summary

- The time complexity class $\mathrm{TIME}(t(n))$ is the collection of languages that are decided by an $\mathcal{O}(t(n))$ time TM
- The time complexity class $\mathrm{NTIME}(t(n))$ is the collection of languages that are decided by an $\mathcal{O}(t(n))$ time NTM
- $\mathrm{TIME}(t(n)) \subset \mathrm{NTIME}(t(n)) \subseteq \mathrm{TIME}(2^{ct(n)})$, for some constant c
- There is an important distinction among models of TMs:
  - There is at most a polynomial difference between the time complexity of problems measured on deterministic single-tape and multitape TMs.
  - There is at most an exponential difference between the time complexity of problems measured on deterministic and nondeterministic TMs