COMPSCI 501: Formal Language Theory

Lecture 21: Time Complexity. Polynomial Time

Marius Minea
marius@cs.umass.edu

University of Massachusetts Amherst

18 March 2019

Questions for Today

- Refine decidable problems into time classes
- Asymptotic complexity: "usual" computers vs. Turing machines
- Single vs. Multi-tape vs. Nondeterministic: what matters?
- Why polynomial as special case ?

Time Complexity

Def. Let M be a deterministic TM that halts on all inputs.
The **time complexity** (*running time*) of M is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps for M to halt on any input of length n.

This corresponds to *worst-case* analysis.

Big-O

Def. Let $f, g : \mathbb{N} \to \mathbb{R}^+$. We say $f(n) = O(g(n))$ if
$\exists c\, \exists n_0\, \forall n \geq n_0 \,.\, f(n) \leq cg(n) \qquad c, n_0 \in \mathbb{N}^+$

$g(n)$ is an **asymptotic upper bound** for $f(n)$
$f(n)$ grows **no faster** than $g(n)$

Ignore multiplicative constants and lower-order terms:

$2n^2 - 5n + 4$ is $O(n^2)$

Sometimes used in exponents:
$f(n) = 2^{O(n)}$ means $f(n) = O(2^{cn})$ for some c.

Small-o

Def. Let $f, g : \mathbb{N} \to \mathbb{R}^+$. We say $f(n) = o(g(n))$ if

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

Equivalent: $\forall c\, \exists n_0\, \forall n \geq n_0 \,.\, f(n) < cg(n)$

$f(n)$ grows *asymptotically slower* than $g(n)$.

Examples:

$\log n = o(n^d)$ for any $d > 0$

$\log \log n = o(\log n)$

$n \log n = o(n \log^2 n)$

$n^d = o(r^n)$ for all $d > 0, r > 1$.

Example: Recognizing $0^k 1^k$

Variant 1:
1. scan tape, reject if 0 right of a 1
2. while both 0s and 1s remain
3. cross off a 0 and a 1
4. if no 0s and 1s remain, *accept*, else *reject*

Complexity? $n + 2$ tape scans, $O(n)$ length $\Rightarrow O(n^2)$.

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

Recognizing $0^k 1^k$ is in TIME(n^2). Surprising? Can we do better?

Faster: Recognizing 0^k1^k

Variant 2 (base-2 counting):
1. scan tape, reject if 0 right of a 1
2. while both 0s and 1s remain
3. if total number is odd, *reject*
4. cross out every other 0 and every other 1
5. if no 0s and 1s remain, *accept*, else *reject*

$2 + \lceil \log n \rceil$ iterations, $O(n)$ each $\Rightarrow O(n \log n)$.

Equivalent: convert 0/1 count to base 2 in $O(n \log n)$, compare

Recognizing 0^k1^k: linear-time with two tapes

Variant 3: two tapes
1. scan tape, reject if 0 right of a 1
2. scan until first 1, copy zeroes to tape 2
3. scan ones on tape 1. For each 1, cross out one 0 on tape 2
4. if too many 0s or 1s, *reject*, else *accept*

Decidability vs. *time complexity*

All computation models decide same class of languages (Church-Turing thesis)

But choice of model affects time complexity
e.g., what can we do in linear time?

Theorem (won't prove): If a language is recognized in $o(n \log n)$ by a one-tape TM, it must be regular!

Single-tape vs. multi-tape

Theorem: Let t be function with $t(n) \geq n$. Then every $t(n)$ multitape TM M has an equivalent $O(t^2(n))$ single-tape TM S.

Proof: Recall simulation of multi-tape TM with single-tape TM:
store tapes consecutively, with markers on heads
repeat
scan all heads to determine next move
update contents and head positions
(possibly shift tape portion right to extend)

Need to bound length of scan:
M does $t(n)$ steps $\Rightarrow$ each tape has length $\leq t(n)$
$\Rightarrow S$ has tape length $\leq kt(n) = O(t(n))$ (k constant)

Each move of S does $\leq k$ tape shifts $\Rightarrow O(t(n))$.

Simulation time: $O(n)$ steps to arrange tape, $t(n)$ steps of $O(t(n))$.
Since $t(n) \geq n$, we get $O(t^2(n))$.

Nondeterministic Turing Machines

Def.. Let N be a nondeterministic TM that is a decider.
The **running time** of N is a 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 any input of length n.

Theorem: Let t be function with $t(n) \geq n$.
Then every $t(n)$ nondeterministic TM N has an equivalent $2^{O(t(n))}$ deterministic single-tape TM D.

Proof. Recall construction that simulates N with D.
Let b be the max. branching factor in the computation tree of N
$\Rightarrow$ tree has at most $b^{t(n)}$ leaves, $O(b^{t(n)})$ nodes.

Each node reached in $\leq t(n)$ steps, so running time is $O(t(n)b^{t(n)}) = 2^{O(t(n))}$.

D has 3 tapes, a single-tape TM (at most) squares complexity:
$(2^{O(t(n))})^2 = 2^{2O(t(n)} = 2^{O(t(n))}$.

The Class P

We've already disregarded constant factors.

Multitape $\to$ single tape conversion a good argument to disregard *polynomial* differences

All (reasonable) *deterministic* computational models are polynomially equivalent.

Def. **P** is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine

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

We'll assume reasonable encodings of input.

We'll *not* consider unary encodings, since they are exponentially larger than encodings in any other base.

Examples: Euclidean Algorithm

gcd(x, y)
1. while $y \neq 0$
2. $x = x \mod y$
3. exchange x and y
4. output x

How many iterations?

if $y \leq x/2$, then $x \mod y < y \leq x/2$
if $y > x/2$, then $x \mod y = x - y < x/2$
$\Rightarrow$ every other iteration cuts x by at least half.

$O(\log x)$ iterations; taking modulo is polynomial $\Rightarrow$ polynomial
can prove $O(\log^2 x)$

Examples: CFL are in P

Theorem: Every context-free language is a member of P.

Recall decidability proof:
  construct Chomsky Normal Form
  take all derivations with $2n - 1$ steps.

Not good enough (number of derivations exponential).

Solution: *dynamic programming*.

$table(i, j)$ stores set of variables that can generate $w_i w_{i+1} \ldots w_j$

$S \to \varepsilon$: special case
initialize $table(i, i)$ with variables $A \to w_i$
for increasing difference $j - i$
  for k from i to $j - 1$
    for all rules $A \to BC$
      if $B \in table(i, k)$ and $C \in table(k + 1, j)$
        add A to $table(i, j)$