Intro to Theory of Computation

CS 332

LECTURE 20

Last time

- Computation history method
- Review, test.

Today

- Recursion theorem
- Complexity theory

Sofya Raskhodnikova

Recursion Theorem

Making TMs that can obtain their own descriptions

with applications to ~~computer~~ TM viruses

A TM P_w that prints w

There is a computable function q that on input w outputs $\langle P_w \rangle$, where P_w is a TM that prints w.

P_w=``Erase input.

1. Print w and halt.''

TM computing q:

``On input w,

1. Print $\langle P_w \rangle$ and halt.''

CS 332 TM S that prints $\langle S \rangle$

Theorem. There is a TM S that erases input, prints $\langle S \rangle$ and halts.

Proof:

- $q(w) = \langle P_w \rangle$

S =``Erase input.
1. Run TM A.
2. Run TM B.
3. Halt.''

A=$P_{\langle B \rangle}$

B =``On input $\langle M \rangle$, where M is a TM,
1. Compute $\langle P_{\langle M \rangle} \rangle = q(\langle M \rangle)$.
2. Construct TM S':

   S′ =``Erase input.
   1. Run TM $P_{\langle M \rangle}$.
   2. Run TM M.
   3. Halt.''
3. Output $\langle S' \rangle$ and halt.''

In English

- Write this sentence.

- Write two copies of the following, the second one in quotes:

``Write two copies of the following, the second

one in quotes:’’

CS 332

Recursion Theorem

If there is a TM T that computes a function $t(w, \langle M \rangle)$ then there is a TM R that computes $r(w) = t(w, \langle R \rangle)$.

Punchline: ``*Obtain your own description*'' is a valid step in an algorithmic description of a TM.

R =``On input *w*,

1. Place # after *w*.
2. Run TM A.
3. Run TM B.
4. Run TM T.''

A=$P'_{\langle B,T \rangle}$

CS 332

Recursion Theorem

If there is a TM T that computes a function $t(w, \langle M \rangle)$ then there is a TM R that computes $r(w) = t(w, \langle R \rangle)$.

R =``On input w,
1. Place # after w.
2. Run TM A.
3. Run TM B.
4. Run TM T.''

A=$P'_{\langle B,T \rangle}$

B =``On input w#$\langle M_{1,} M_2 \rangle$,
where M_1, M_2 are TMs,
1. Compute $\langle P'_{\langle M_1, M_2 \rangle} \rangle = q'(\langle M_1, M_2 \rangle)$.
2. Construct TM R':

   $\boldsymbol{R'}$ =``On input w,
   1. Place # after w.
   2. Run TM $P'_{\langle M_1, M_2 \rangle}$.
   3. Run TM M_1.
   4. Run TM M_2''
3. Output w#$\langle R' \rangle$ and halt.''

CS 332 Recursion Theorem

Punchline: *``Obtain your own description''* is a valid step in an algorithmic description of a TM.

Application of recursion theorem

- Give an alternative proof that A_{TM} is undecidable.

(on the board)

Application of recursion theorem

- A TM M is **minimal** if there is no TM equivalent to M that has a shorter description than $\langle M \rangle$.
- $MIN_{TM} = \{\langle M \rangle \mid M$ is minimal TM$\}$.
- Show that MIN_{TM} is not Turing-recognizable.

(on the board)

Complexity Theory

Already learned

- Automata theory
- Computability theory

Last unit: complexity theory

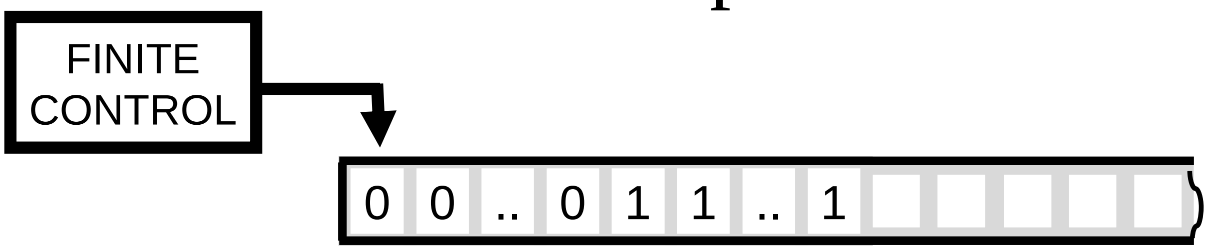
First topic: time complexity

- Measuring complexity (as in Algorithms)
- Asymptotic notation (as in Algorithms)
- Relationships between models

How much time/memory needed to decide a language?

Example: Consider $A = \{0^m 1^m | m \geq 0\}$.

- Time needed for 1-tape TM?



- M_1=“
  1. Scan input and reject if it is not of the form 0^*1^*.
  2. Repeat while both 0s and 1s remain on the tape:
  3. Cross off one 0 and one 1
  4. **Accept** if no 0s and no 1s left; otherwise **reject**.”

Running time analysis

If M is a TM and $f: \mathbb{N} \to \mathbb{N}$ then

"M runs in time $f(n)$" means

for **every** input $w \in \Sigma^*$ of length n,

M on w halts within $f(n)$ steps

- Focus on worst case:
  - upper bound on running time for all inputs of given length
- Exact time depends on the computer
  - instead measure **asymptotic growth**

Asymptotic notation

O-notation (upper bounds):

$f(n) = O(g(n))$ means

there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

Example: $2n^2 = O(n^3)$ $(c = 2, n_0 = 1)$

functions, not values

Asymptotic Notation

- **One-sided equality:** $T(n) = O(f(n))$.
  - Not transitive:
    - $f(n) = 5n^3$; $g(n) = 3n^2$
    - $f(n) = O(n^3)$ and $g(n) = O(n^3)$
    - but $f(n) \neq g(n)$.
  - Alternative notation: $f(n) \in O(g(n))$.

Examples

- $10^6 n^3 + 2n^2 - n + 10 =$ $O(n^3)$
- $\sqrt{n} + \log n =$ $O(\sqrt{n})$
- $n(\log n + \sqrt{n}) =$ $O(n\sqrt{n})$
- $n =$ $O(n)$, also $O(n^2)$

CS 332 Ω-notation (lower bounds)

O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$f(n) = \Omega(g(n))$ means

there exist constants $c > 0$, $n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0$.

Example: $\sqrt{n} = \Omega(\log n)$ ($c = 1$, $n_0 = 16$)

Ω-notation (lower bounds)

- **Be careful:** "Any comparison-based sorting algorithm requires at least O(n log n) comparisons."
  - Meaningless!
  - Use Ω for lower bounds.

CS 332

Θ-notation (tight bounds)

$\Theta(g(n))$ means both $O(g(n))$ and $\Omega(g(n))$

Example: $\frac{1}{2}n^2 - 2n = \Theta(n^2)$

Polynomials are simple:

$$a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0 = \Theta(n^d)$$

o-notation and ω-notation

O-notation and Ω-notation are like $\leq$ and $\geq$.
o-notation and ω-notation are like $<$ and $>$.

$f(n) = \mathrm{o}(g(n))$ means

for **every** constant $c > 0$,
there exists a constants $n_0 > 0$
such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

Example: $2n^2 = o(n^3)$ $(n_0 = 2/c)$

CS 332

Overview

Notation	… means …	Think…	Example	$\lim\limits_{n \leftarrow \infty} \frac{f(n)}{g(n)}$
$f(n)=O(g(n))$	$\exists\ c>0, n_0>0, \forall\ n > n_0$: $0 \leq f(n) < cg(n)$	Upper bound	$100n^2$ $= O(n^3)$	If it exists, it is $< \infty$
$f(n)=\Omega(g(n))$	$\exists c>0, n_0>0, \forall\ n > n_0$: $0 \leq cg(n) < f(n)$	Lower bound	2^n $= \Omega(n^{100})$	If it exists, it is > 0
$f(n)=\Theta(g(n))$	both of the above: $f=\Omega(g)$ **and** $f = O(g)$	Tight bound	$\log(n!)$ $= \Theta(n \log n)$	If it exists, it is > 0 and $< \infty$
$f(n)=o(g(n))$	$\forall\ c>0, \exists\ n_0>0, \forall\ n > n_0$: $0 \leq f(n) < cg(n)$	Strict upper bound	$n^2 = o(2^n)$	Limit exists, $=0$
$f(n)=\omega(g(n))$	$\forall\ c>0, \exists n_0>0, \forall\ n > n_0$: $0 \leq cg(n) < f(n)$	Strict lower bound	n^2 $= \omega(\log n)$	Limit exists, $=\infty$

Common Functions: Asymptotic Bounds

- **Polynomials.** $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.
- **Logarithms.** $\log_a n = \Theta(\log_b n)$ for all constants $a, b > 0$.

  can avoid specifying the base

  log grows slower than every polynomial

  For every $x > 0$, $\log n = o(n^x)$.

  Every exponential grows faster than every polynomial

- **Exponentials.** For all $r > 1$ and all $d > 0$, $n^d = o(r^n)$.
- **Factorial.** $n! = n(n-1)\cdots 1$.

  By Sterling's formula,

$$n! = (\sqrt{2\pi n})\left(\frac{n}{e}\right)^n (1 + o(1)) = 2^{\Theta(n \log n)}$$

Sort by asymptotic order of growth

a) $n \log n$

b) $\sqrt{n}$

c) $\log n$

d) n^2

e) 2^n

f) n

g) $\log \log n$

h) $n!$

i) $n^{1,000,000}$

j) $n^{1/\log(n)}$

k) $\log(n!)$

l) $\binom{n}{2}$

m) 2^{n^2}

n) 2^{2^n}

CS 332

Time complexity classes

TIME($f(n)$) is a class of languages.

$A \in$ **TIME($f(n)$)** means that

some 1-tape TM M

that runs in time O($f(n)$) decides A.

The class P

P is the class of languages decidable in polynomial time on a ***deterministic*** 1-tape TM:

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

- The same class even if we substitute another reasonable deterministic model.
- Roughly the class of problems realistically solvable on a computer.