Intro to Theory of Computation

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 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."

11/13/2018
Theorem. There is a TM S that erases input, prints \( \langle S \rangle \) and halts.

Proof:

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

\( S = \text{``Erase input.} \)
1. Run TM A.
2. Run TM B.
3. Halt.’’

\( A = P_{\langle B \rangle} \)

\( B = \text{``On input } \langle M \rangle, \text{ where } M \text{ is a TM,} \)
1. Compute \( \langle P_{\langle M \rangle} \rangle = q(\langle M \rangle) \).
2. Construct TM \( S' \):
   \( S' = \text{``Erase input.} \)
   1. Run TM \( P_{\langle M \rangle} \).
   2. Run TM \( M \).
   3. Halt.’’
3. Output \( \langle S' \rangle \) and halt.’’
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:’’
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}**
If there is a TM $T$ that computes a function $t(w, \langle M \rangle)$ then there is a TM $R$ that computes $r(w) = t(w, \langle R \rangle)$.

$R = \text{``On input } w, \text{ where } M_1, M_2 \text{ are TMs,} \text{''} \begin{align*} 
1. \text{ Place } \# \text{ after } w. \\
2. \text{ Run TM } A. \\
3. \text{ Run TM } B. \\
4. \text{ Run TM } T. \end{align*}$

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

$B = \text{``On input } w\#\langle M_1, M_2 \rangle, \text{''} \begin{align*} 
1. \text{ Compute } P'_{\langle M_1, M_2 \rangle} = q'(\langle M_1, M_2 \rangle). \\
2. \text{ Construct TM } R': \\
R' = \text{``On input } w, \text{''} \begin{align*} 
1. \text{ Place } \# \text{ after } w. \\
2. \text{ Run TM } P'_{\langle M_1, M_2 \rangle}. \\
3. \text{ Run TM } M_1. \\
4. \text{ Run TM } M_2. \end{align*}$

3. Output $w\#\langle R' \rangle$ and halt.''}
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 $\langle M \rangle$.
- $MIN_{TM} = \{ \langle M \rangle \mid M \text{ is minimal TM} \}$.
- Show that $MIN_{TM}$ is not Turing-recognizable.

(on the board)
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 $A = \{0^m1^m | m \geq 0\}$.

- Time needed for 1-tape TM?

- $M_1 = \{1. \text{ Scan input and reject if it is not of the form } 0^*1^*. \\
2. \text{ Repeat while both 0s and 1s remain on the tape:} \\
3. \text{ Cross off one 0 and one 1} \\
4. \text{ 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

\( f(n) = O(g(n)) \) means there exist constants \( c > 0 \), \( n_0 > 0 \) such that \( 0 \leq f(n) \leq cg(n) \) for all \( n \geq n_0 \).

**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

- \(10^6 n^3 + 2n^2 - n + 10 = O(n^3)\)
- \(\sqrt{n} + \log n = O(\sqrt{n})\)
- \(n (\log n + \sqrt{n}) = O(n\sqrt{n})\)
- \(n = O(n), \text{ also } O(n^2)\)
\(\Omega\)-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 \(0 \leq cg(n) \leq f(n)\) for all \(n \geq n_0\).

**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 $\Omega$ for lower bounds.
Θ(\(g(n)\)) means both \(O(g(n))\) and \(Ω(g(n))\)

**Example:** \(\frac{1}{2}n^2 - 2n = Θ(n^2)\)

Polynomials are simple:
\[a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0 = Θ(n^d)\]
O-notation and Ω-notation are like ≤ and ≥.

o-notation and ω-notation are like < and >.

\[ f(n) = o(g(n)) \] 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:** \( 2n^2 = o(n^3) \) \( (n_0 = 2/c) \)
# Overview

<table>
<thead>
<tr>
<th>Notation</th>
<th>... means …</th>
<th>Think…</th>
<th>Example</th>
<th>$\lim \frac{f(n)}{g(n)}_{n\rightarrow \infty}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(n) = O(g(n))$</td>
<td>$\exists c \gt 0, n_0 \gt 0, \forall n &gt; n_0 :$ \n$0 \leq f(n) &lt; cg(n)$</td>
<td>Upper bound</td>
<td>$100n^2$ \n$= O(n^3)$</td>
<td>If it exists, it is $\lt 1$</td>
</tr>
<tr>
<td>$f(n) = \Omega(g(n))$</td>
<td>$\exists c \gt 0, n_0 \gt 0, \forall n &gt; n_0 :$ \n$0 \leq cg(n) &lt; f(n)$</td>
<td>Lower bound</td>
<td>$n^{100}$ \n$= \Omega(2^n)$</td>
<td>If it exists, it is $\gt 0$</td>
</tr>
<tr>
<td>$f(n) = \Theta(g(n))$</td>
<td>both of the above: \n$f = \Omega(g)$ and $f = O(g)$</td>
<td>Tight bound</td>
<td>$\log(n!)$ \n$= \Theta(n \log n)$</td>
<td>If it exists, it is $\gt 0$ and $\lt 1$</td>
</tr>
<tr>
<td>$f(n) = o(g(n))$</td>
<td>$\forall c \gt 0, \exists n_0 \gt 0, \forall n &gt; n_0 :$ \n$0 \leq f(n) &lt; cg(n)$</td>
<td>Strict upper bound</td>
<td>$n^2 = o(2^n)$ \n$= 0$</td>
<td>Limit exists, $= 0$</td>
</tr>
<tr>
<td>$f(n) = \omega(g(n))$</td>
<td>$\forall c \gt 0, \exists n_0 \gt 0, \forall n &gt; n_0 :$ \n$0 \leq cg(n) &lt; f(n)$</td>
<td>Strict lower bound</td>
<td>$n^2$ \n$= \omega(\log n)$</td>
<td>Limit exists, $= 1$</td>
</tr>
</tbody>
</table>
Common Functions: Asymptotic Bounds

- **Polynomials.** \(a_0 + a_1n + \ldots + a_d n^d\) is \(\Theta(n^d)\) if \(a_d > 0\).

- **Logarithms.** \(\log_a n = \Theta(\log_b n)\) for all constants \(a, b > 0\).
   
   For every \(x > 0\), \(\log n = o(n^x)\).

- **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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a)</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>b)</td>
<td>$\sqrt{n}$</td>
</tr>
<tr>
<td>c)</td>
<td>$\log n$</td>
</tr>
<tr>
<td>d)</td>
<td>$n^2$</td>
</tr>
<tr>
<td>e)</td>
<td>$2^n$</td>
</tr>
<tr>
<td>f)</td>
<td>$n$</td>
</tr>
<tr>
<td>g)</td>
<td>$\log \log n$</td>
</tr>
<tr>
<td>h)</td>
<td>$n!$</td>
</tr>
<tr>
<td>i)</td>
<td>$n^{1,000,000}$</td>
</tr>
<tr>
<td>j)</td>
<td>$n^{1/\log(n)}$</td>
</tr>
<tr>
<td>k)</td>
<td>$\log(n!)$</td>
</tr>
<tr>
<td>l)</td>
<td>$\binom{n}{2}$</td>
</tr>
<tr>
<td>m)</td>
<td>$2^{n^2}$</td>
</tr>
<tr>
<td>n)</td>
<td>$2^{2^n}$</td>
</tr>
</tbody>
</table>

11/13/2018
TIME($f(n)$) is a class of languages. 

$A \in \text{TIME}(f(n))$ means that some 1-tape TM M that runs in time $O(f(n))$ decides A.
The class $P$ is the class of languages decidable in polynomial time on a deterministic 1-tape TM:

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

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