cs3102: Theory of Computation

Class 23:
P=NP?

Spring 2010
University of Virginia
David Evans

PS6: due April 27

Menu
Exam 2
Algorithm Analysis Recap
Complexity Classes $P$ and $NP$
P = NP?

Questions on Both Exam 1 and Exam 2

• Understanding that the empty language is regular, and that grammars can describe languages in any class inside the grammar type (Exam 1: 2a; Exam 2: 1a)
• Understanding that if a model has finite memory it cannot be more powerful than DFA (Exam 1: LeakyPDR; Exam 2: Fading Tape)
• Understanding how to define a machine type that corresponds to a strange circle-crossing class (Exam 1: 3; Exam 2: 3a)

Since many people still had troubles with these, you should not be surprised if they appear on the final also!

Exam 2, Problem 1d

$UVA-WINS = \{ y \mid y \text{ is a four-digit string representing a year between 2000 and 2999 and UVa wins an NCAA soccer championship (Men's or Women's) in that year} \}$

What if this question is reused in Term 1G 3000's cs3063141592502: Theory of Computation class?

Theoretical Questions are Universal

Ali G: Can the biggest computer today multiply $9 \times 9 \times 9 \times 9 \times \ldots \times 9$?
Ali G: Will a computer ever be able to multiply $9 \times 9 \times 9 \times 9 \times \ldots \times 9$?

Not Ali G: Is the multiplication problem computable?
Not Ali G: Is the runtime of the multiplication problem in $O(n \log n 2^{6(\log^* n)})$?

Before 2007: We don’t know.
After 2007: Yes (Furer’s algorithm)

Today: We don’t know.
Sometime in the future: ?

Should the answer to a theoretical question depend on when you ask it?
**UVA-WINS**

**Definition of a regular language:**
A language is a **regular language** if there is some DFA that recognizes it.

To prove a language is regular we just need to know there is some DFA that recognizes it. We don’t need to know which DFA!

Since the language is finite, there is definitely some DFA that recognizes it. We could list the set of possible DFAs today, of which one is correct. We don’t know which one is correct until 2999.

**Church-Turing Thesis**

The answer to “Is language X decidable?” is the same for all reasonable computing models.
The answer to “Is language X recognizable?” is the same for all reasonable computing models.

The simple Turing machine model is very robust:
reasonable changes to the model do not increase the set of languages that are decidable or recognizable.

What’s a “reasonable” change?

One that doesn’t increase the set of languages that are decidable or the set of languages that are recognizable.

Yes, it’s a circular definition: remember it’s a “thesis” not a “theorem”.

**A “Reasonable” Change**

Exam 2 Problem 4: Janus Machine

$L(TM) \subseteq L(Janus\ Machine)$:
Janus Machine can simulate any TM
Make the alpha-head follow the same rules as the TM.
Make the zeta-head always write whatever symbol it reads and move L.

$L(Janus\ Machine) \subseteq L(TM)$:
TM can simulate any Janus Machine

**Simulating JM with TM**

To simulate one JM step: \(\delta(q, x_a, x_z) \rightarrow (q', y_a, d_a, y_z, d_z)\):

1. Search tape to find the symbol \(\hat{x}\) or \(\hat{y}\). Let \(x_a = x\).
2. Search tape to find the symbol \(\hat{y}\) or \(\hat{y}\). Let \(x_z = y\).
3. Match the rule using the simulated state, \(q, x_a, x_z\) as inputs.
4. Search the tape to find a symbol that contains a \(\hat{x}\) or \(\hat{y}\), replace it with \(y\). Transition to a bookkeeping state, and move \(d_a\). Replace the symbol \(x\) in that square with \(\hat{x}\) (if it contains \(\hat{x}\), replace it with \(\hat{x}\)).
5. Search the tape to find a symbol that contains a \(\hat{y}\) or \(\hat{y}\), replace it with \(y\). Transition to a bookkeeping state, and move \(d_z\). Replace the symbol \(x\) in that square with \(\hat{x}\) (if it contains \(\hat{x}\), replace it with \(\hat{x}\)).
6. Update the simulated state to \(q'\).

What if both heads are on the same square?

**A “Reasonable” Change?**

Sure, it’s a subset of \(T-R\), so doesn’t increase TM power.
"Reasonable" Attempts that Fail

Too weak to include any undecidable:
- Limit length of tape
- Limit number of steps

Too strong to exclude any decidable:
- Limit number of states (only need 2 for UTM)
- Limit number of tape symbols (only need 2 for UTM)

Force Reject on $1\Sigma^*$

$R_{ITM} = \text{A Turing Machine where there is the start state has a transition on input 1 to } q_{reject}$.

Infinitely many TM-D languages no $R_{ITM}$ can recognize:
Any language that includes any string that starts with a 1.

Can recognize some undecidable languages:
$A1_{TM} = \{ \langle M, w \rangle | M \text{ is a TM description that starts with a 0 and } M \text{ accepts } w \}$

Asymptotic Operators Recap

**Big-O:** grow no faster than $f$

$f \in O(g)$ means there exist $c, n_0 \in \mathbb{R}^+$ such that

$$\forall n \geq n_0 : f(n) \leq cg(n)$$

**Omega ($\Omega$):** grow no slower than $f$

$f \in \Omega(g)$ means there exist $c, n_0 \in \mathbb{R}^+$ such that

$$\forall n \geq n_0 : f(n) \geq cg(n)$$

**Theta ($\Theta$):** grow as fast as $f$

$f \in \Theta(g)$ iff $f \in O(g)$ and $f \in \Omega(g)$.

Acing cs2XXX Algorithm Analysis

```c
int gaussSum (int m) {
    int sum = 0;
    for (int i = 1; i <= m; i++) {
        sum = sum + i;
    }
    return sum;
}
```

Question: running time of X in big-O notation?

Always correct answer: $O(BB(n))$

Question: running time of Java/C++/C program X in $\Theta$ notation?

Nearly always correct answer: $\Theta(1)$

Constant Time Classes

What are $O(1)$, $\Theta(1)$, and $\Omega(1)$?

What is the 1?

Input to $O$ is $N \rightarrow \mathbb{R}^+$. 1 means the function:

$f(n) = 1$ for all $n \in N$.

What is $O(1)$?

$f \in O(g)$ means there exist $c, n_0 \in \mathbb{R}^+$ such that

$$\forall n \geq n_0 : f(n) \leq cg(n)$$

So, $f \in O(1)$ if there exist $c, n_0 \in \mathbb{R}^+$ such that

$$\forall n \geq n_0 : f(n) \leq c$$

So, $O(1)$ includes only functions whose output does not grow at all.
What is $\Omega(1)$?

$f \in \Omega(g)$ means there exist $c, n_0 \in \mathbb{R}^+$ such that

$$\forall n \geq n_0 : f(n) \geq cg(n)$$

So, $f \in \Omega(1)$ if there exist $c, n_0 \in \mathbb{R}^+$ such that

$$\forall n \geq n_0 : f(n) \geq c$$

So, $\Omega(1)$ includes all functions in $\mathcal{N} \rightarrow \mathbb{R}^+$.

What is $\Theta(1)$?

$f \in \Theta(g)$ iff $f \in O(g)$ and $f \in \Omega(g)$.

$$\Theta(1) = O(1) \cap \Omega(1) = O(1).$$

By convention, (most) people write $O(1)$ to mean constant time.

Non-Constant Time Java Program?

```java
public class RealList{
  private Object node;
  private RealList next;
  ...
  public Boolean contains(Object val) {
    if (this.node == null) return false;
    if (this.node == val) return true;
    else return this.next.contains(val);
  }
}
```

Running time scales with number of elements in list, so is in $\Theta(n)$.

Note: this is not the case for Java library LinkedList and other collection types. They are all bounded by Integer.MAX_INT.

Robustness of TM Model

- Computability: all reasonable variations on TMs have the same or less computing power
  - TM can simulate Janus Machine, multi-tape machine, nondeterministic TM, DFA, PDA, etc.
- Complexity: are our complexity classes robust to reasonable machine models?

Are there problems that are in $O(1)$ for a Janus machine but in $\Omega(n)$ for a regular Turing Machine?

So what should a (non-evil) CS2150 TA do?

Can we find a way to define “$\Theta$” so the “desired” answers are correct?

$f \in \Theta(g)$ iff $f \in O(g)$ and $f \in \Omega(g)$.

$$f \in O(g) \text{ means there exist } c, n_0 \in \mathbb{R}^+, n_0 < 100 \text{ such that }$$

$$\forall n_0 \leq n \leq 2^{31} : f(n) \leq cg(n)$$

Still doesn’t work...for the right choice for $c$, everything is still in $O(1)$.

Challenge problem: find a precise definition of $\Theta$ such that the expected $\text{MAX}_n$ for cs2150 algorithm analysis questions are actually correct. (Not sure this is possible)
Non-Robustness of $\Theta(n)$

$END1 = \{ w1 | w \in \Sigma^* \}$

Regular Turing Machine: in $\Omega(n)$:
A TM requires at least $n$ steps to get to the end of the input and check if it's a $1$.

Janus Machine: in $O(1)$:
Zeta-head takes one step (move $L$) to see end of input, and one step to check it's a $1$.

Theory is about Big Questions

If little tweaks to our model change the answers, we might as well focus on answering (Ali G’s) practical questions about real systems and specific problem instances instead.

Making things Robustier?

- Find a more robust computing model
  - Church-Turing thesis says all mechanical models are equivalent (computing power) to a TM
  - But, this doesn’t mean there might not be more robust models for complexity
- Make the complexity classes bigger
  - Define a complexity class big enough so the little tweaks to TMs do not change the answers

The asymptotic operators abstract away issues like how long a step takes and how many tape symbols you have, but not issues like number of tapes or heads.

Complexity Class $P$

$P = \bigcup_{k \in \mathbb{N}} TIME(N^k)$

$P$ is the class of languages that can be decided in Polynomial Time on a deterministic, single-tape Turing machine.

Classes in $P$

- a) $TIME(N^2)$
- b) $TIME(O(N^7))$
- c) $TIME(O(2^N))$
- d) Class of languages that can be decided in Polynomial Time by a 2-tape TM
- e) Class of languages that can be decided in Polynomial Time by a nondeterministic TM

Unknown! This is the $P = \text{NP}$ question!

Deterministic TM

$(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

- $Q$ finite set of states
- $\Sigma$ input alphabet, finite set of symbols (cannot include $\cup$)
- $\Gamma$ tape alphabet, finite set of symbols (includes $\Sigma$ and $\cup$)
- $\delta$ transition function: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{ L, R \}$
- $q_0$ start state, $q_0 \in Q$
- $q_{\text{accept}}$ accepting state, $q_{\text{accept}} \in Q$
- $q_{\text{reject}}$ rejecting state, $q_{\text{reject}} \in Q$, $q_{\text{reject}} \neq q_{\text{accept}}$

What needs to change to define a Nondeterministic TM?
Nondeterministic TM

For a given state and input symbol, there may be many possible transitions.

Deterministic TM

$\delta^* : \Gamma^* \times Q \times \Gamma^* \rightarrow \Gamma^* \times Q \times \Gamma^*$

$\forall u, v \in \Gamma^*, a, b \in \Gamma, q \in Q$:

if $q \in \{q_{\text{accept}}, q_{\text{reject}}\}$: $\delta^*(u, q_F, v) = (u, q_F, v)$

else:

if $\delta(q, b) = (q, c, L)$: $\delta^*(ua, q, bv) = \delta^*(u, q_F, av)$

if $\delta(q, b) = (q, c, R)$: $\delta^*(ua, q, q_F) = \delta^*(u, q_F, av)$

A Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ accepts a string $w \in \Sigma^*$ iff

$\delta^*(\varepsilon, q_0, w) = (\gamma_L, q_{\text{accept}}, \gamma_R)$ for some $\gamma_L, \gamma_R \in \Gamma^*$.

What needs to change to define a Nondeterministic TM?

Deterministic

0 1 0 1 1 0 ...

q0

1 1 0 1 1 0 ...

q1

1 0 0 1 1 0 ...

q2

Non-deterministic

0 1 0 1 1 0 ...

q0

1 1 1 1 1 0 ...

q1

1 0 1 1 1 0 ...

q2

Tries all possible transitions; accepts if any path leads to accepting state.

Computability:

Is NDTM more powerful than DTM?

No! We can simulate a NDTM with a DTM. Use a tape to keep track of which paths to try (breadth-first search, not depth first!)

See Sipser Theorem 3.16 for details.

Complexity:

Is a NDTM faster than a DTM?

- It is definitely “faster” with non-robust complexity classes: an NDTM can decide some languages in fewer steps than DTM.
- Is it “faster” with robust Class P?

- Polynomial time: languages that can be decided by a deterministic TM in $\Theta(N^k)$ steps.

$$P = \bigcup_{k \in N} TIME(N^k)$$

- Nondeterministic Polynomial time: languages that can be decided by a non-deterministic TM in $\Theta(N^k)$ steps.

$$NP = \bigcup_{k \in N} TIME_{NDTM}(N^k)$$

Is there any language that a NDTM can recognize in polynomial time that is not in P?
P = NP?

Option 1: \( P \subseteq NP \)
Option 2: \( P = NP \)

Theological Question

If God exists (and is omnipotent), could she compute anything regular people cannot compute?

**Yes:** \( P \subseteq NP \)
Being able to always guess right when given a decision makes you more powerful than having to try both.

**No:** \( P = NP \)
Being able to always guess right when given a decision does not make you more powerful than having to try both.

Charge

- Next week:
  - How to make progress towards answer \( P=NP \)?
  - Examples of the hardest problems in \( NP \)
  - Non-theological implications of each possible answer

- PS6 is due Tuesday, April 27

Return Exam 2 and PS5