BU CS 332 – Theory of Computation

Lecture 25:

- Final review

Reading:

Sipser Ch 7.1-8.2, 9.1

Mark Bun

December 9, 2021

Final Topics

Everything from Midterms 1 and 2

- Midterm 1 topics: DFAs, NFAs, regular expressions, distinguishing set method
  - (more detail in lecture 8 notes)
- Midterm 2 topics: Turing machines, TM variants, Church-Turing thesis, decidable languages, countable and uncountable sets, undecidability, reductions, unrecognizability
  - (more detail in lecture 16 notes)

Mapping Reducibility (5.3)

- Understand the definition of a computable function
- Understand the definition of a mapping reduction
- Know how to use mapping reductions to prove decidability, undecidability, recognizability, and unrecognizability

$A \leq_m B$

B recognizable $\Rightarrow$ A recognizable

A unrecognizable $\Rightarrow$ B unrecognizable

Time ~~and~~ Complexity (7.1)

- Asymptotic notation: Big-Oh, little-oh
- Know the definition of running time for a TM and of time complexity classes (TIME / NTIME)
- Understand how to simulate multi-tape TMs and NTMs using single-tape TMs and know how to analyze the running time overhead

P and NP (7.2, 7.3)

- Know the definitions of P and NP as time complexity classes
- Know how to analyze the running time of algorithms to show that languages are in P / NP
- Understand the verifier interpretation of NP and why it is equivalent to the NTM definition
- Know how to construct verifiers and analyze their runtime

NP-Completeness (7.4, 7.5)

- Know the definition of poly-time reducibility
- Understand the definitions of NP-hardness and NP-completeness
- Understand the statement of the Cook-Levin theorem (don't need to know its proof) SAT is NP-complete
- Understand several canonical NP-complete problems and the relevant reductions: SAT, 3SAT, CLIQUE, INDEPENDENT-SET, VERTEX-COVER, HAMPATH, SUBSET-SUM

Space Complexity (8.1, 8.2)

- Know the definition of running space for a TM and of space complexity classes (SPACE / NSPACE)
- Understand the known relationships between space complexity classes and time complexity classes
- Understand the statement of Savitch's Theorem
- Know the definitions of PSPACE and NPSPACE and the relationship between them (i.e., PSPACE = NPSPACE)

$\text{TIME}(f(n)) \subseteq \text{SPACE}(f(n)) \subseteq \text{TIME}(2^{O(f(n))})$

$\text{NSPACE}(f(n)) \subseteq \text{SPACE}((f(n))^2)$

Hierarchy Theorems (9.1)

- Formal statements of time and space hierarchy theorems and how to apply them
- How to use hierarchy theorems to prove statements like $\mathrm{P} \neq \mathrm{EXP}$

E.g. THT $\quad f(n) = o\left(\frac{g(n)}{\log g(n)}\right) \Rightarrow$

$\mathrm{TIME}(f(n)) \subseteq \mathrm{TIME}(g(n))$ and

$\mathrm{TIME}(g(n)) \not\subseteq \mathrm{TIME}(f(n))$

[More compactly: $\mathrm{TIME}(f(n)) \subsetneq \mathrm{TIME}(g(n))$]

Things we didn't get to talk about

- Additional classes between NP and PSPACE (polynomial hierarchy)
- Logarithmic space
- Relativization and the limits of diagonalization
- Boolean circuits
- Randomized algorithms / complexity classes
- Interactive and probabilistic proof systems
- Complexity of counting

https://cs-people.bu.edu/mbun/courses/535_F20/

Theory and Algorithms Courses after 332

- Algorithms
  - CS 530/630 (Advanced algorithms)
  - CS 531 (Optimization algorithms)
  - CS 537 (Randomized algorithms)
- Complexity
  - CS 535 (Complexity theory)
- Cryptography
  - CS 538 (Foundations of crypto)
- Topics (CS 591)

  E.g., Privacy in machine learning, algorithms and society, sublinear algorithms, new developments in theory of computing, communication complexity

Algorithms and Theory Research Group

- https://www.bu.edu/cs/research/theory/

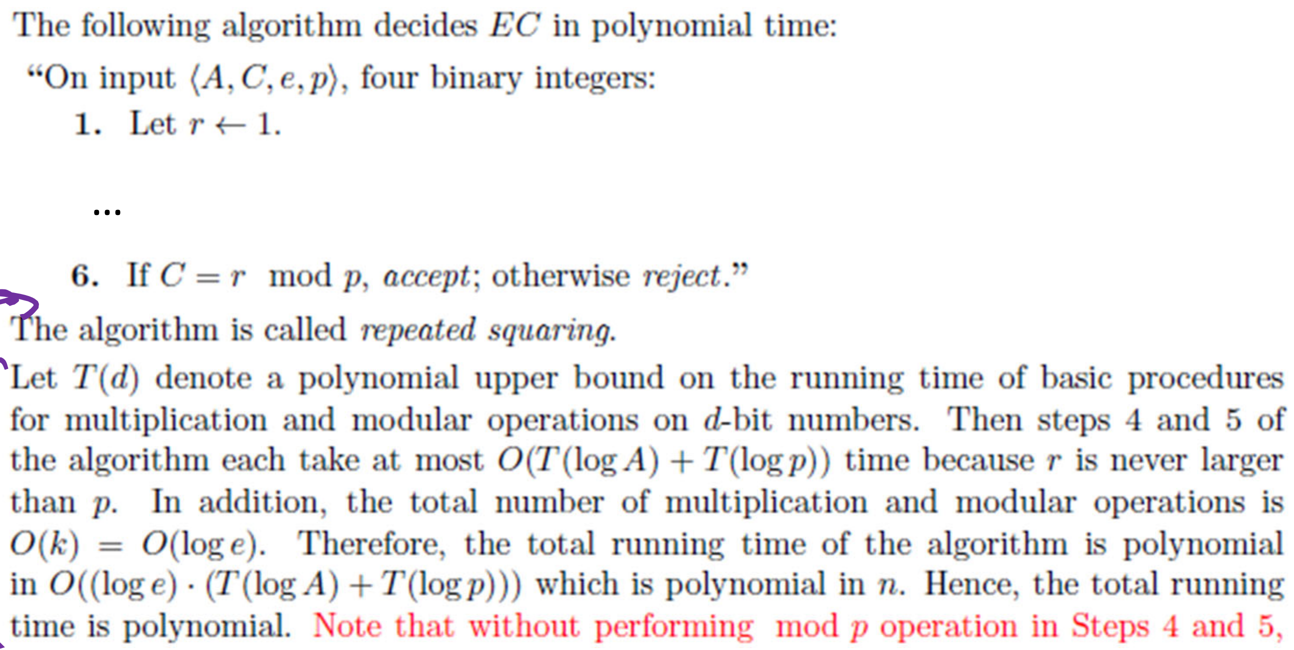
- Weekly seminar: Mondays at 11
  https://www.bu.edu/cs/algorithms-and-theory-seminar/

  Great way to learn about research in theory of computation!

Tips for Preparing Exam Solutions

Designing (nondeterministic) time/space-bounded deciders

High level description



The following algorithm decides EC in polynomial time:

```
"On input ⟨A, C, e, p⟩, four binary integers:
    1. Let r ← 1.

    ...

    6. If C = r mod p, accept; otherwise reject."
```

The algorithm is called *repeated squaring*.

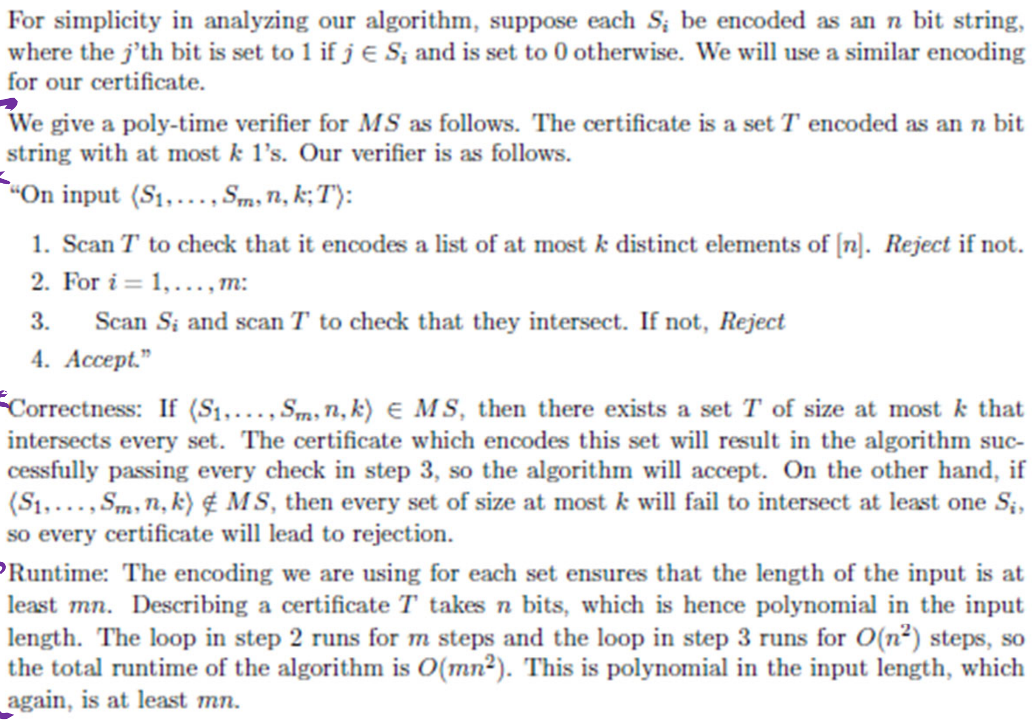
Explanation of correctness should go here (I was bad)

Let $T(d)$ denote a polynomial upper bound on the running time of basic procedures for multiplication and modular operations on d-bit numbers. Then steps 4 and 5 of the algorithm each take at most $O(T(\log A) + T(\log p))$ time because r is never larger than p. In addition, the total number of multiplication and modular operations is $O(k) = O(\log e)$. Therefore, the total running time of the algorithm is polynomial in $O((\log e) \cdot (T(\log A) + T(\log p)))$ which is polynomial in n. Hence, the total running time is polynomial. Note that without performing mod p operation in Steps 4 and 5,

Explanation of runtime

- Key components: High-level description of algorithm, explanation of correctness, analysis of running time and/or space usage

Designing NP verifiers



For simplicity in analyzing our algorithm, suppose each S_i be encoded as an n bit string, where the j'th bit is set to 1 if $j \in S_i$ and is set to 0 otherwise. We will use a similar encoding for our certificate.

Certificate

We give a poly-time verifier for MS as follows. The certificate is a set T encoded as an n bit string with at most k 1's. Our verifier is as follows.

Verifier

```
"On input $\langle S_1, \ldots, S_m, n, k; T \rangle$:
  1. Scan $T$ to check that it encodes a list of at most $k$ distinct elements of $[n]$. Reject if not.
  2. For $i = 1, \ldots, m$:
  3.     Scan $S_i$ and scan $T$ to check that they intersect. If not, Reject
  4. Accept."
```

Explain correctness

Correctness: If $\langle S_1, \ldots, S_m, n, k \rangle \in MS$, then there exists a set T of size at most k that intersects every set. The certificate which encodes this set will result in the algorithm successfully passing every check in step 3, so the algorithm will accept. On the other hand, if $\langle S_1, \ldots, S_m, n, k \rangle \notin MS$, then every set of size at most k will fail to intersect at least one S_i, so every certificate will lead to rejection.

Explain poly runtime

Runtime: The encoding we are using for each set ensures that the length of the input is at least mn. Describing a certificate T takes n bits, which is hence polynomial in the input length. The loop in step 2 runs for m steps and the loop in step 3 runs for $O(n^2)$ steps, so the total runtime of the algorithm is $O(mn^2)$. This is polynomial in the input length, which again, is at least mn.

- Key components: Description of certificate, high-level description of algorithm, explanation of correctness, analysis of running time

NP-completeness proofs

To show a language L is NP-complete:

1) Show L is in NP (follow guidelines from previous two slides)

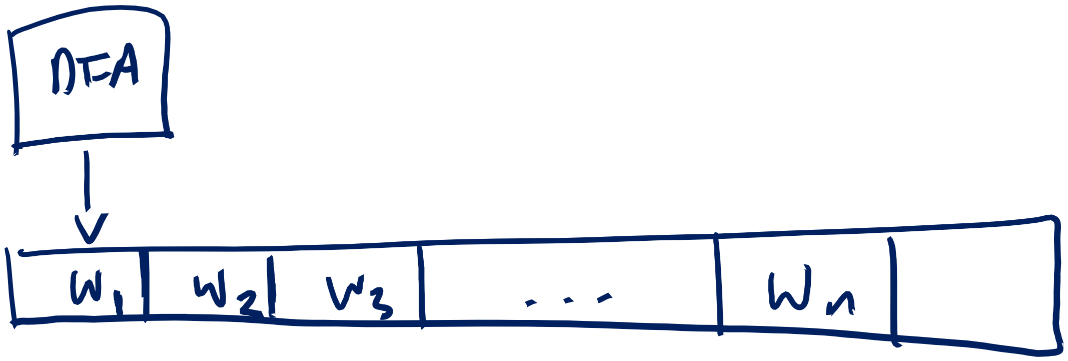
2) Show L is NP-hard (usually) by giving a poly-time reduction $A \leq_p L$ for some NP-complete language A

   - High-level description of algorithm computing reduction
   - Explanation of correctness: Why is $w \in A$ iff $f(w) \in L$ for your reduction f?
   - Analysis of running time

Practice Problems

What is the time complexity of regular languages?

If L is regular, $\exists$ DFA recognizing L



Think of a DFA as a special case of a TM that runs in exactly n steps on every input of length n. In other words, runs in linear time n

L is regular $\Rightarrow$ $L \in \text{TIME}(n)$

Thm: $L \in TIME(f(n))$ for $f(n) = o(n \log n)$

$\Rightarrow L$ is regular

Sample Final Problem 2

$A \in NP$, but $\underbrace{A \text{ is not generated by a regex}}$

i.e. A is not regular

E.g. $A = \{0^n 1^n \mid n \geq 0\}$ {know this is not regular}

To show $A \in NP$:

Enough to construct a poly time verifier for A:

Certificate: Number n representing # of 0's and # of 1's

Verifier $V(\langle w ; n \rangle)$:

1) Scan w to check that it starts w/ n 0's
2) keep scanning to check that it ends w/ n 1's
3) Accept iff checks pass.

Can also show $A \in P$ (Fact that $P \subseteq NP$ implies $A \in NP$)

2-tape TM deciding A in deterministic poly-time:

On input w:

1) Check that $w \in L(0^* 1^*)$ // One linear scan

2) Copy all 1's to tape 2

3) Scan tapes 1 and 2 in parallel to check # 0's = # 1's.

b) B is recognizable, but B is not in EXP

$B \in EXP$ means: B is $\boxed{\text{decidable}}$

by a TM running in time

$O(2^{n^k})$ for some const. k

$B = A_{TM}$

Know: A_{TM} is recognizable, but

A_{TM} not decidable $\Rightarrow$ A_{TM} not in EXP

c) $A_{TM} \le_m C$ but C is not recognizable

[Note: If $L \le_m A_{TM}$, then A is recognizable]
But not relevant here

1st Try: $C = \overline{A_{TM}}$ $\quad A_{TM} \le_m \overline{A_{TM}}$

Can't work: Suppose for contradiction $A_{TM} \le_m \overline{A_{TM}}$.

Then $\overline{A_{TM}} \le_m A_{TM}$ (since $A \le_m B \Rightarrow \overline{A} \le_m \overline{B}$)

$\Rightarrow \overline{A_{TM}}$ is recognizable by purple note.

$\Rightarrow A_{TM}$ is recognizable (UTM) and $\overline{A_{TM}}$ is recognizable

$\Rightarrow A_{TM}$ is decidable $\times$.

2nd Thing: $C = EQ_{TM}$

Cool Fact: EQ_{TM} and $\overline{EQ_{TM}}$ are both unrecog

Claim: $A_{TM} \leq_m EQ_{TM}$

Mapping reduction from A_{TM} to EQ_{TM}:

" On input $\langle M, w \rangle$:

1. Construct TM N_1:
   On input x:
      Run M on w, output result
2. Construct TM N_2:
   Accept
3. Output $\langle N_1, N_2 \rangle$ "

Want: M accepts w $\iff L(N_1) = L(N_2)$

M accepts $w \Rightarrow$
$L(N_1) = \Sigma^*$

M does not accept $w \Rightarrow$
$L(N_1) = \emptyset$

$L(N_2) = \Sigma^*$

Use a mapping reduction to show that $ALL_{\mathrm{TM}} = \{\langle M\rangle | M \text{ is a TM and } L(M) = \Sigma^*\}$ is co-unrecognizable

Use a mapping reduction to show that $ALL_{\text{TM}} = \{\langle M\rangle | M \text{ is a TM and } L(M) = \Sigma^*\}$ is unrecognizable

Give examples of the following languages: 1) A language in P. 2) A decidable language that is not in P. 3) A language for which it is unknown whether it is in P.

Give an example of a problem that is solvable in polynomial-time, but which is not in P

Let $L = \{\langle w_1, w_2\rangle | \exists \text{ strings } x, y, z \text{ such that } w_1 = xyz \text{ and } w_2 = xy^R z\}$. Show that $L \in \mathrm{P}$.

Which of the following operations is P closed under? Union, concatenation, star, intersection, complement.

Prove that $LPATH = \{\langle G, s, t, k\rangle | G \text{ is an directed graph containing a simple path from } s \text{ to } t \text{ of length} \geq k\}$ is in NP

Prove that $LPATH$ is NP-hard

Which of the following operations is NP closed under? Union, concatenation, star, intersection, complement.

Which of the following statements are true?

- $SPACE(2^n) = SPACE(2^{n+1})$ Equal

$2^n \leq 2^{n+1}$ means $SPACE(2^n) \subseteq SPACE(2^{n+1})$

Is $SPACE(2^{n+1}) \subseteq SPACE(2^n)$?
"$SPACE(2 \cdot 2^n) = SPACE(2^n)$"

Recall: $SPACE(f(n)) = \{A \mid A$ is decidable in space $O(f(n))\}$

- $SPACE(2^n) = SPACE(3^n)$ Not equal

$SPACE(2^n) \subseteq SPACE(3^n)$, but $SPACE(3^n) \not\subseteq SPACE(2^n)$

$f(n) = 2^n$ $\quad 2^n = o(3^n)$ b/c $\lim_{n \to \infty} \frac{2^n}{3^n} = \lim_{n\to\infty} \left(\frac{2}{3}\right)^n = 0$.
$g(n) = 3^n$

SHT: If $f(n) = o(g(n))$ then $SPACE(f(n)) \subsetneq SPACE(g(n))$

- $NSPACE(n^2) = SPACE(n^5)$ Not Equal

$NSPACE(n^2) \subseteq SPACE(n^4)$ (Savitch)

$\subsetneq SPACE(n^5)$ (SHT + $n^4 = o(n^5)$)

Savitch: $NSPACE(f(n)) \subseteq SPACE(f(n)^2)$