Space ... and Beyond

Wednesday, May 11, 2022
Announcements

• HW 12 due tonight 11:59pm EST
  • Last HW!

• Last lecture!
Previously: **NP-Completeness**

**DEFINITION**

A language $B$ is **NP-complete** if it satisfies two conditions:

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

These are the “hardest” problems (in NP) to solve.
**NP-Completeness vs NP-Hardness**

**Definition**

A language $B$ is **NP-complete** if it satisfies two conditions:

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

“NP-Complete” = in NP + “NP-Hard”

So a language can be NP-hard but not NP-complete!
Flashback: The Halting Problem

\[ HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

Thm: \( HALT_{TM} \) is undecidable

Proof, by contradiction:

\begin{itemize}
  \item Assume \( HALT_{TM} \) has decider \( R \); use it to create decider for \( A_{TM} \):
  \begin{itemize}
    \item ...
  \end{itemize}
\end{itemize}

\begin{itemize}
  \item But \( A_{TM} \) is undecidable and has no decider!
Flashback: The Halting Problem

\[ \text{HALT}_{\text{TM}} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

Thm: \text{HALT}_{\text{TM}} \text{ is undecidable}

Proof, by contradiction:

• Assume \text{HALT}_{\text{TM}} has decider \( R \); use it to create decider for \( A_{\text{TM}} \):

\[ S = \text{“On input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ and a string } w: \]

1. Run TM \( R \) on input \( \langle M, w \rangle \).
2. If \( R \) rejects, \textit{reject}. \( \leftarrow \) This means \( M \) loops on input \( w \)
3. If \( R \) accepts, simulate \( M \) on \( w \) until it halts. \( \leftarrow \) This step always halts
4. If \( M \) has accepted, \textit{accept}; if \( M \) has rejected, \textit{reject.”}
Flashback: The Halting Problem

\[ HALT_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

Thm: \( HALT_{TM} \) is undecidable

Proof, by contradiction:

• Assume \( HALT_{TM} \) has decider \( R \); use it to create decider for \( A_{TM} \):

\[ S = \text{“On input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ and a string } w:} \]

  1. Run TM \( R \) on input \( \langle M, w \rangle \).
  2. If \( R \) rejects, reject.
  3. If \( R \) accepts, simulate \( M \) on \( w \) until it halts.
  4. If \( M \) has accepted, accept; if \( M \) has rejected, reject.”

• But \( A_{TM} \) is undecidable!
  • I.e., this decider that we just created cannot exist! So \( HALT_{TM} \) is undecidable.
The Halting Problem is \textbf{NP-Hard}

\[ \text{HALT}_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

Proof: Reduce 3\textit{SAT} to the Halting Problem

(Why does this prove that the Halting Problem is \textbf{NP-hard}?)

Because 3\textit{SAT} is \textbf{NP}-complete! (so every \textbf{NP} problem is poly time reducible to 3\textit{SAT})
The Halting Problem is **NP-Hard**

\[ \text{HALT}_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

**Computable function, from 3SAT → HALT_{TM}:**

On input \( \phi \), a formula in 3cnf:

- **Construct TM** \( M \)
  
  \[ M = \text{on input } \phi \]

  - Try all assignments
    - If any satisfy \( \phi \), then **accept**
    - When all assignments have been tried, start over

- **Output** \( \langle M, \phi \rangle \)

\[ (x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (x_3 \lor \overline{x_5} \lor x_6) \land (x_3 \lor \overline{x_6} \lor x_4) \]

\[ (x_1 \lor \overline{x_2} \lor x_3) \land (x_3 \lor \overline{x_5} \lor x_6) \land (x_3 \lor \overline{x_6} \lor x_4) \]

\[ \text{HALT}_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

⇒ If \( \phi \) has a satisfying assignment, then \( M \) halts on \( \phi \)

⇐ If \( \phi \) has no satisfying assignment, then \( M \) loops on \( \phi \)
**Definition**

A language $B$ is *NP-complete* if it satisfies two conditions:

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

So a language can satisfy condition #2 but not condition #1.

But can a language satisfy condition #1 but not condition #2?

Yes, every language in P ...

... unless $P = NP$
NP-Completeness vs NP-Hardness

Is there any problem definitely outside of here?
Space ...
**Flashback: Dynamic Programming Example**

- **Chomsky Grammar** $G$:
  - $S \rightarrow AB \mid BC$
  - $A \rightarrow BA \mid a$
  - $B \rightarrow CC \mid b$
  - $C \rightarrow AB \mid a$

- **Example string**: $baaba$

- Store every *partial string* and their generating variables in a *table*

<table>
<thead>
<tr>
<th>Substring start char</th>
<th>Substring end char</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td>vars for “b”</td>
<td>vars for “ba”</td>
<td>vars for “baa”</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td>vars for “a”</td>
<td>vars for “aa”</td>
<td>vars for “aab”</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We are gaining time ...  
... by spending more space!
Space Complexity, Formally

**DEFINITION**

Let \( M \) be a deterministic Turing machine that halts on all inputs. The **space complexity** of \( M \) is the function \( f : \mathbb{N} \rightarrow \mathbb{N} \), where \( f(n) \) is the maximum number of tape cells that \( M \) scans on any input of length \( n \). If the space complexity of \( M \) is \( f(n) \), we also say that \( M \) runs in space \( f(n) \).

If \( M \) is a nondeterministic Turing machine wherein all branches halt on all inputs, we define its space complexity \( f(n) \) to be the maximum number of tape cells that \( M \) scans on any branch of its computation for any input of length \( n \).
Space Complexity Classes

**DEFINITION**

Let $f: \mathcal{N} \rightarrow \mathcal{R}^+$ be a function. The *space complexity classes*, $\text{SPACE}(f(n))$ and $\text{NSPACE}(f(n))$, are defined as follows.

$\text{SPACE}(f(n)) = \{ L \mid L \text{ is a language decided by an } O(f(n)) \text{ space deterministic Turing machine} \}$.

$\text{NSPACE}(f(n)) = \{ L \mid L \text{ is a language decided by an } O(f(n)) \text{ space nondeterministic Turing machine} \}$.

**Compare:**

Let $t: \mathcal{N} \rightarrow \mathcal{R}^+$ be a function. Define the *time complexity class*, $\text{TIME}(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

$\text{NTIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine} \}$.
Example: \( SAT \) Space Usage

\[
\text{\textcolor{blue}{2}^{0(m)}} \text{ exponential time machine}
\]

\[
M_1 = \text{“On input } \langle \phi \rangle, \text{ where } \phi \text{ is a Boolean formula:}
\]
1. For each truth assignment to the variables \( x_1, \ldots, x_m \) of \( \phi \):
2. Evaluate \( \phi \) on that truth assignment.
3. If \( \phi \) ever evaluated to 1, accept; if not, reject.”

\[
\text{\textcolor{green}{Each loop iteration requires } O(m) \text{ space}}
\]

\[
\text{\textcolor{green}{But the space is re-used on each loop! (nothing is stored from the last loop)}}
\]

\[
\text{\textcolor{green}{So the entire machine only needs } O(m) \text{ space!}}
\]

\[
\text{\textcolor{blue}{Space is “more powerful” than time.}}
\]

\[
\text{\textcolor{green}{SAT is in } O(m) \text{ space complexity class!}}
\]

\[
\text{\textcolor{green}{SAT} = \{ \langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}}
\]
Example: Nondeterministic Space Usage

\[ \text{ALL}_{\text{NFA}} = \{ \langle A \rangle | A \text{ is an NFA and } L(A) = \Sigma^* \} \]

Nondeterministic decider for \( \text{ALL}_{\text{NFA}} \) (accepts NFAs that reject something)

\[ N = \text{“On input } \langle M \rangle, \text{ where } M \text{ is an NFA:} \]

1. Place a marker on the start state of the NFA.
2. Repeat \( 2^q \) times, where \( q \) is the number of states of \( M \):
   - Nondeterministically select an input symbol and change the positions of the markers on \( M \)'s states to simulate reading that symbol.
3. Accept if stages 2 and 3 reveal some string that \( M \) rejects; that is, if at some point none of the markers lie on accept states of \( M \). Otherwise, reject.”

\[ q \text{ states } = 2^q \text{ possible combinations (so exponential time)} \]

Additional, need a counter to count to \( 2^q \); this requires \( \log(2^q) = q \) extra space

But each loop uses only \( O(q) \) space!

So the whole machine runs in (nondeterministic) linear \( O(q) \) space!
Facts About Time vs Space (for Deciders)

**TIME → SPACE**
- If a decider runs in time $t(n)$, then its maximum space usage is ...
- $t(n)$
- ... because it can add at most 1 tape cell per step

**SPACE → TIME**
- If a decider runs in space $f(n)$, then its maximum time usage is ...
- $(|\Gamma| + |Q|)^{f(n)} = 2^{df(n)}$
- ... because that’s the number of possible configurations
- (and a decider cannot repeat a configuration)
Flashback: Deterministic vs Non-Det. Time

• If a non-deterministic TM runs in: $t(n)$ time
• Then an equivalent deterministic TM runs in: $2^{O(t(n))}$
  • Exponentially slower

What about space?
Deterministic vs Non-Det. Space

**THEOREM**

Savitch’s theorem  For any function \( f: \mathbb{N} \rightarrow \mathbb{R}^+ \), where \( f(n) \geq n \),
\[ \text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)). \]

- If a non-deterministic TM runs in: \( f(n) \) space
- Then an equivalent deterministic TM runs in: \( f^2(n) \) space
  - **Exponentially** Only \textbf{Quadratically} slower!
**Flashback:** Non-det. TM $\rightarrow$ Deterministic TM

- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - 1-2
    - 1-1-1

$$b^t(n) = 2^{2^{O(t(n))}}$$

**Diagram:**
- Nondeterministic computation
- Max height (longest path) $t(n)$
- Max # of paths $b = \text{branching per level}$
- Accept

319
Flashback: Non-det \(\rightarrow\) Deterministic TM: Space

- **Input Tape**: \(n\) space
- **Simulation Tape**: \(t(n)\) space
- **Address Tape**: \(2^{O((n))}\) (exponential) space

3 tapes
Let $N$ be an NTM deciding language $A$ in space $f(n)$
- This means a single path could use $f(n)$ space
- That path could take $2^{df(n)}$ steps
  - (That’s the possible ways to fill the space)
  - Each step could be a non-deterministic branch that must be saved
- So naively tracking these branches requires $2^{df(n)}$ space!

• Instead, let’s “divide and conquer” to reduce space!
“Divide and Conquer” TM Config Sequences

• Want to check whether:

  \[ C_{\text{start}} \xrightarrow{2^{O(f(n))} \text{ (possibly branching) steps}} C_{\text{accept}} \]

  Remembering the branch at every step costs exponential space

  So long as we save the intermediate config

  Each split must remember a “\( c_m \)” config = \( O(f(n)) \) space

  \[ \log(2^{O(f(n))}) = O(f(n)) \]

• Instead, we check whether:

  \[ C_{\text{start}} \xrightarrow{2^{O(f(n))}/2 \text{ steps}} C_m \xrightarrow{2^{O(f(n))}/2 \text{ steps}} C_{\text{accept}} \]

  Remembering these steps costs half the space ...

  ... and we can reuse that space to check the second half

  \[ \text{Total: } O(f(n)) \times O(f(n)) = O(f^2(n)) \text{ space} \]

  (Savitch’s Thm)

• Keep dividing ...

  \[ C_{\text{start}} \xrightarrow{} \xrightarrow{} \xrightarrow{} C_{\text{accept}} \]
Formally: A “Yielding” Algorithm

\begin{align*}
\text{CANYIELD} &= \text{“On input } c_1, c_2, \text{ and } t:\n1. \text{ If } t = 1, \text{ then test directly whether } c_1 = c_2 \text{ or whether } c_1 \text{ yields } c_2 \text{ in one step according to the rules of } N. \text{ Accept if either test succeeds; reject if both fail.}
2. \text{ If } t > 1, \text{ then for each configuration } c_m \text{ of } N \text{ using space } f(n):
3. \text{ Run } \text{CANYIELD}(c_1, c_m, \frac{t}{2}).
4. \text{ Run } \text{CANYIELD}(c_m, c_2, \frac{t}{2}).
5. \text{ If steps 3 and 4 both accept, then accept.}
6. \text{ If haven’t yet accepted, reject.”}
\end{align*}
Savitch’s Theorem: Proof

• Let $N$ be an NTM deciding language $A$ in space $f(n)$
• Construct equivalent deterministic TM $M$ using $O(f^2(n))$ space:

$$M = "\text{On input } w:\n1. \text{ Output the result of CANYIELD}(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})\."$$

• $c_{\text{start}} = \text{start configuration of } N$
• $c_{\text{accept}} = \text{new accepting config where all } N\text{'s accepting configs go}$

Extra $d$ constant depends on size of tape alphabet
PSPACE

**Definition**

PSPACE is the class of languages that are decidable in polynomial space on a deterministic Turing machine. In other words,

\[ \text{PSPACE} = \bigcup_{k} \text{SPACE}(n^k). \]
**NPSPACE**

Analogous to P and NP for time complexity

**DEFINITION**

NPSPACE is the class of languages that are decidable in polynomial space on a deterministic Turing machine. In other words,

\[
\text{NPSPACE} = \bigcup_{k} \text{NSPACE}(n^k).
\]

But \( P \subseteq \text{PSPACE} \) and \( NP \subseteq \text{NPSPACE} \)
- Because each step can use at most one extra tape cell
- But space can be re-used
Flashback: Does $P = NP$?

Proving $P \neq NP$ is hard because how do you prove an algorithm doesn’t have a poly time algorithm? (in general it’s hard to prove that something doesn’t exist)
PSPACE = NPSPACE?

- **PSPACE**: langs decidable in poly space on deterministic TM
- **NPSPACE**: langs decidable in poly space on nondeterministic TM

**Theorem**: PSPACE = NPSPACE !!!

**Proof**: By Savitch’s Theorem!

---

**Savitch’s theorem** For any function \( f : \mathbb{N} \rightarrow \mathbb{R}^+ \), where \( f(n) \geq n \), 
\[ \text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)). \]
Space vs Time

• \( P \subseteq \text{PSPACE} \) and \( \text{NP} \subseteq \text{NPSPACE} \)
  • Because each step can use at most one extra tape cell
  • And space can be re-used

• \( \text{PSPACE} \subseteq \text{EXPTIME} \)
  • Because an \( f(n) \) space TM has \( 2^{O(f(n))} \) possible configurations
  • And a halting TM cannot repeat a configuration

• We already know \( P \subseteq \text{NP} \) and \( \text{PSPACE} = \text{NPSPACE} \) ... so:

\[
P \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME}
\]
Space vs Time: Conjecture

 Researchers believe these are all completely contained within each other

 But this is an open conjecture!

 Only known result so far is: \( P \subseteq \text{EXPTIME} \)
(this means some problems provably have no poly time algorithm!)

\[ P \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME} \]
Last Quiz 5/11

In gradescope