NP

Monday, April 26, 2021
Announcements

• HW10 due Tues 4/27 11:59pm EST

• HW11 out soon
  • Due Tues 5/4 11:59pm EST

• **Reminder:** Submitted HW must be in your own words
  • Not “your own words”: Submitting answers from the internet
  • Not “your own words”: Changing variables / rearranging sentences
  • Suggestion: Looking into “clean room” design
Last Time: Polynomial Time ($P$)

**Definition 7.12**

$P$ is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

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

- Roughly corresponds to **solvable vs unsolvable** problems:
  - Problems in $P$ = “solvable”
  - Problems outside $P$ = “unsolvable”
Today: Search vs Verification

- **Search** problems are often **unsolvable**
- But, **verification** of search results is usually **solvable**

**Examples**

- Factoring
  - **Unsolvable**: Find factors of 8633
  - **Solvable**: Verify 89 and 97 are factors of 8633

- Passwords
  - **Unsolvable**: Find my umb.edu password
  - **Solvable**: Verify whether my umb.edu password is ... "correct horse battery staple"
Last Time: The \textit{PATH} Problem

\[
\text{PATH} = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t \}\]

- The \textbf{search} problem:
  - Exponential time (brute force) algorithm \((n^n)\):
    - Check all possible paths and see if any connects \(s\) and \(t\)
  - Polynomial time algorithm:
    - Do a breadth-first search (roughly), marking "seen" nodes as we go

\begin{proof}
A polynomial time algorithm \(M\) for \textit{PATH} operates as follows.

\(M = \text{"On input } \langle G, s, t \rangle, \text{ where } G \text{ is a directed graph with nodes } s \text{ and } t:\)

1. Place a mark on node \(s\).
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of \(G\). If an edge \((a, b)\) is found going from a marked node \(a\) to an unmarked node \(b\), mark node \(b\).
4. If \(t\) is marked, accept. Otherwise, reject."
\end{proof}
Verifying a \textit{PATH} \\

\textit{PATH} = \{(G, s, t)| G \text{ is a directed graph that has a directed path from } s \text{ to } t\}

• **The verification problem:**
  • Given some path \( p \) in \( G \), check that it is a path from \( s \) to \( t \)
  • Let \( m = \text{longest possible path} = \# \text{ edges in } G \)
  • **Verifier** \( V = \text{On input } <G, s, t, p>, \text{ where } p \text{ is some set of edges:} \)
    1. Check some edge in \( p \) has “from” node \( s \); mark and set it as “current” edge
      • Max steps = \( O(m) \)
    2. Loop: While there remains unmarked edges in \( p \):
      a) Find the “next” edge in \( p \), whose “from” node is the “to” node of “current” edge
      b) If found, then mark that edge and set it as “current”, else reject
      • Each loop: Max steps \( O(m) \)
      • \# loops: at most \( m \) times
      • Total looping time = \( O(m^2) \)
    3. Check “current” edge has “to” node \( t \); if yes accept, else reject

• Total time = \( O(m) + O(m^2) = O(m^2) = \text{polynomial in } m \)

\textbf{NOTE: extra argument } p \textbf{PATH can be verified in polynomial time}
Verifiers, Formally

\[ \text{PATH} = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t \} \]

**Definition 7.18**

A verifier for a language \( A \) is an algorithm \( V \), where

\[ A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \}. \]

We measure the time of a verifier only in terms of the length of \( w \), so a polynomial time verifier runs in polynomial time in the length of \( w \). A language \( A \) is **polynomially verifiable** if it has a polynomial time verifier.

- **NOTE**: a cert \( c \) must be at most length \( n^k \), where \( n = \text{length of } w \)
  - Why?
- So \( \text{PATH} \) is polynomially verifiable
The **HAMPATH** Problem

- A Hamiltonian path goes through every node in the graph

**The Search problem:**
- Exponential time (brute force) algorithm:
  - Check all possible paths and see if any connect \( s \) and \( t \) using all nodes
- Polynomial time algorithm:
  - We don’t know if there is one!!!

**The Verification problem:**
- Still \( \mathcal{O}(m^2) \)!
- \( HAMPATH \) is polynomially verifiable, but **not** polynomially decidable
The class **NP**

- *PATH* is in **NP**, and **P**
- *HAMPATH* is in **NP**, but not **P**
**NP** = **Nondeterministic polynomial time**

**Definition 7.19**

NP is the class of languages that have polynomial time verifiers.

**Theorem 7.20**

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

=> If a lang \( L \) is in NP, then we know it has a poly time verifier \( V \)

- **Need to**: Create NTM deciding \( L \): on input \( w = \)
  - Nondeterministically run \( V \) with \( w \) and all possible certificates \( c \)

<= If \( L \) has NTM decider \( N \),

- **Need to**: show \( L \) is in NP, ie it has polytime verifier \( V \): on input \(<w, c> = \)
  - Convert \( N \) to deterministic TM, and run it on \( w \), but take only one computation path
  - Let certificate \( c \) dictate which computation path to follow
P vs NP

**Definition 7.7**
Let \( t : \mathbb{N} \rightarrow \mathbb{R}^+ \) 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.

**Definition 7.12**
P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,
\[
P = \bigcup_k \text{TIME}(n^k).
\]

**Definition 7.21**
\( \text{NTIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine} \}. \)

**Corollary 7.22**
\[
\text{NP} = \bigcup_k \text{NTIME}(n^k).
\]
More **NP** Problems

- **CLIQUE** $= \{\langle G, k \rangle \mid G$ is an undirected graph with a $k$-clique$\}$
  - A clique is a subgraph where every two nodes are connected
  - A $k$-clique contains $k$ nodes

- **SUBSET-SUM** $= \{\langle S, t \rangle \mid S = \{x_1, \ldots, x_k\}$, and for some\n  \[\{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \sum y_i = t\} \}$
Theorem: **CLIQUE is in NP**

\[ \text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique} \} \]

**Proof Idea**  The clique is the certificate.

**Proof**  The following is a **verifier** \( V \) for **CLIQUE**.

\( V = \text{"On input } \langle \langle G, k \rangle, c \rangle:\)

1. Test whether \( c \) is a subgraph with \( k \) nodes in \( G \). \( O(k) \)
2. Test whether \( G \) contains all edges connecting nodes in \( c \). \( O(k^2) \)
3. If both pass, accept; otherwise, reject.”

---

**Definition 7.18**  A **verifier** for a language \( A \) is an algorithm \( V \), where

\[ A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \} \]

We measure the time of a verifier only in terms of the length of \( w \), so a **polynomial time verifier** runs in polynomial time in the length of \( w \). A language \( A \) is **polynomially verifiable** if it has a polynomial time verifier.

**Definition 7.19**  \( \text{NP} \) is the class of languages that have polynomial time verifiers.
Proof 2: \textbf{CLIQUE} is in NP

\[ \text{CLIQUE} = \{ \langle G, k \rangle | \text{G is an undirected graph with a } k\text{-clique} \} \]

\[ N = \text{“On input } \langle G, k \rangle \text{, where } G \text{ is a graph:} \]
\[ \begin{align*}
1. & \quad \text{Nondeterministically select a subset } c \text{ of } k \text{ nodes of } G. \\
2. & \quad \text{Test whether } G \text{ contains all edges connecting nodes in } c. \\
3. & \quad \text{If yes, } \text{accept}; \text{ otherwise, } \text{reject.”} \\
\end{align*} \]

To prove a lang } L \text{ is in NP, create either a:}
- **Deterministic** poly time verifier
- **Nondeterministic** poly time decider

\text{THEOREM 7.20}

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.
More **NP** Problems

- **CLIQUE** = \{⟨G, k⟩\} | G is an undirected graph with a k-clique
  - A clique is a subgraph where every two nodes are connected
  - A k-clique contains k nodes

- **SUBSET-SUM** = \{⟨S, t⟩\} \ S = \{x_1, \ldots, x_k\}, and for some
  \ \{y_1, \ldots, y_l\} ≤ \{x_1, \ldots, x_k\}, we have \(\Sigma y_i = t\)
  - Some subset of a set of numbers S must sum to some total t
  - e.g., \{4, 11, 16, 21, 27\}, 25 \in **SUBSET-SUM**

\[\text{Graph Image}\]
Theorem: \textit{SUBSET-SUM is in NP}

\[ \text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{ x_1, \ldots, x_k \}, \text{ and for some } \{ y_1, \ldots, y_l \} \subseteq \{ x_1, \ldots, x_k \}, \text{ we have } \sum y_i = t \} \]

**Proof Idea** The subset is the certificate.

To prove a lang is in NP, create either:
- Deterministic poly time verifier
- Nondeterministic poly time decider

**Proof** The following is a verifier \( V \) for \textit{SUBSET-SUM}.

\( V = \) “On input \( \langle \langle S, t \rangle, c \rangle \):
1. Test whether \( c \) is a collection of numbers that sum to \( t \).
2. Test whether \( S \) contains all the numbers in \( c \).
3. If both pass, \textit{accept}; otherwise, \textit{reject}.”

**Alternative Proof** We can also prove this theorem by giving a nondeterministic polynomial time Turing machine for \textit{SUBSET-SUM} as follows.

\( N = \) “On input \( \langle S, t \rangle \):
1. Nondeterministically select a subset \( c \) of the numbers in \( S \).
2. Test whether \( c \) is a collection of numbers that sum to \( t \).
3. If the test passes, \textit{accept}; otherwise, \textit{reject.”}
COMPOSITES = \{x \mid x = pq, \text{ for integers } p, q > 1\}

• A composite number is not prime

• COMPOSITES is polynomially verifiable
  • i.e., it’s in NP
  • i.e., factorability is in NP

• A certificate could be:
  • Some factor that is not 1

• Checking existence of factors (or not, i.e., testing primality) ...
  • ... is also poly time
  • But only discovered recently (2002)
Question: Does $P = NP$?

One of the greatest unsolved mysteries in science

$P = NP$  

$P \text{ vs } \text{NP}$

$\text{PATH}$

$\text{P}$

$\text{CLIQUE}$

$\text{HAMPATH}$

$\text{COMPOSITES}$

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)
Implications if $P = NP$

- Every problem with a “brute force” solution also has an efficient solution

- I.e., “unsolvable” problems are “solvable”

**BAD:**
- Cryptography needs unsolvable problems
- Near perfect AI learning, recognition

**GOOD:** Optimization problems are solved
- Overcrowding or world hunger solved?
- Abundant energy resources?
Progress on whether $P = NP$?

• Some, but still not close

The Status of the P Versus NP Problem

By Lance Fortnow
Communications of the ACM, September 2009, Vol. 52 No. 9, Pages 78-86
10.1145/1562164.1562186

• One important concept discovered:
  • NP-Completeness (next time)
Next time: NP-Completeness

**Definition 7.34**
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 $B$.

- How does this help the $P = NP$ problem?

**Theorem 7.35**
If $B$ is NP-complete and $B \in P$, then $P = NP$
Check-in Quiz 4/26
On gradescope