Who doesn't like niche NP jokes?

AN ENGINEER, A PHYSICIST, AND A MATHEMATICIAN ARE ROOMMATES AND ARE MOVING TO A NEW PLACE.

AS THE MOVER PULLS UP, THE MATHEMATICIAN WORRIES THERE ISN'T ENOUGH ROOM.

THE MOVER REASSURES THEM. I BEEN AT THIS 30 YEARS. I CAN LOOK AT ANY AMOUNT OF STUFF AND INSTANTLY TELL YA IF IT CAN FIT IN THE MOVING BINS.

THE ENGINEER SAYS...
IT'S OBVIOUS IT CAN FIT. ANYTHING THAT DOESN'T GO IN THE BINS CAN BE TAPED TO THE ROOF.

THE PHYSICIST SAYS...
IT'S OBVIOUS IT CAN FIT. IF IT WERE THE DENSITY OF A NEUTRON STAR, OUR STUFF WOULD BE THE SIZE OF A BASEBALL.

THE MATHEMATICIAN SAYS...
PLEASE DON'T HACK MY EMAIL.

UMB CS420
NP
Monday, April 25, 2022
Announcements

• HW 10 out
  • Due Tuesday 4/26 11:59pm EST

• Hannah Office Hours moved
  • Now Monday 2-3:30pm in-person
  • McCormack, 3rd Floor, Room 0201-33
Last Time: 3 Problems in \( \mathbf{P} \)

• A *Graph* Problem:
  \[
  \text{PATH} = \{ \langle G, s, t \rangle | \text{G is a directed graph that has a directed path from } s \text{ to } t \}\]

• A *Number* Problem:
  \[
  \text{RELPRIME} = \{ \langle x, y \rangle | x \text{ and } y \text{ are relatively prime} \}\]

• A *CFL* Problem:
  Every context-free language is a member of \( \mathbf{P} \)
Search vs Verification

- **Search** problems are often **unsolvable**
- But, **verification** of a search result 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”
The *PATH* Problem

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

- It’s a **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

---

**Proof**

A polynomial time algorithm \(M\) for *PATH* operates as follows.

\(M = \) “On input \(\langle G, s, t \rangle\), where \(G\) is a directed graph with nodes \(s\) 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.”
Verifying a $PATH$

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

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 $V = \text{On input } \langle G, s, t, p \rangle, \text{ where } p \text{ is some set of edges:}$

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 $p$:
   1. Find the “next” edge in $p$, whose “from” node is the “to” node of “current” edge
   2. If found, then mark that edge and set it as “current”, else reject
      • Each loop iteration: $O(m)$
      • # loops: $O(m)$
      • Total looping time $= O(m^2)$

3. Check “current” edge has “to” node $t$; if yes accept, else reject

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

$PATH$ can be verified in polynomial time
Verifiers, Formally

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 $PATH$ is polynomially verifiable
The **HAMPATH** Problem

\[
\text{HAMPATH} = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t \}\]

- 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 \( O(m^2) \)!
- \( \text{HAMPATH} \) is polynomially verifiable, but **not** polynomially decidable.
The class **NP**

**DEFINITION**

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

- **PATH** is in NP, and P
- **HAMPATH** is in NP, but it’s unknown whether it’s in P
\[ \textbf{NP} = \textbf{Nondeterministic polynomial time} \]

\textbf{NP} is the class of languages that have polynomial time verifiers.

**Theorem**

A language is in \( \text{NP} \) iff it is decided by some nondeterministic polynomial time Turing machine.

\[ \Rightarrow \text{If a language is in } \text{NP}, \text{ then it has a non-deterministic poly time decider} \]

- **We know**: If a lang \( L \) is in \( \text{NP} \), then it has a poly time verifier \( V \)
- **Need to**: create NTM deciding \( L \):

  On input \( w = \)
  
  - Nondeterministically run \( V \) with \( w \) and all possible poly length certificates \( c \)

\[ \Leftarrow \text{If a language has a non-deterministic poly time decider, then it is in } \text{NP} \]

- **We know**: \( L \) has NTM decider \( N \)
- **Need to**: show \( L \) is in \( \text{NP} \), i.e., create 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
NP

\[ \text{NP} = \bigcup_k \text{NTIME}(n^k) \]

\[ \text{NTIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine} \}. \]
Let $t: \mathcal{N} \to \mathcal{R}^+$ 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.

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

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

$P = \text{Deterministic polynomial time}$

$\text{NTIME}(t(n)) = \{L | L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}$.

$\text{NP} = \bigcup_k \text{NTIME}(n^k)$

$\text{NP} = \text{Nondeterministic polynomial time}$
More **NP** Problems

- **CLIQUE** = \{\langle G, k \rangle | 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 | S = \{x_1, \ldots, x_k\}, and for some 
  \( \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \) we have \( \sum y_i = t \)\}
Theorem: **CLIQUE** is in NP

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

### Proof Idea
The clique is the certificate.

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

\[ V = \text{“On input } \langle \langle G, k \rangle, c \rangle \text{:} \]

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

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.

### How to prove a language is in NP:
**Proof technique #1: create a verifier**

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

Let \( n = \# \text{ nodes in } G \)

\( c \) is at most \( n \)

For each node in \( c \), check whether it’s in \( G \): \( O(n^2) \)

For each pair of nodes in \( c \), check whether there’s an edge in \( G \): \( O(n^2) \)
Proof 2: **CLIQUE** is in NP

**CLIQUE** = \{ (G, k) | G is an undirected graph with a k-clique \}

\[ N = \text{"On input } (G, k), \text{ where } G \text{ is a graph:}
1. \text{Nondeterministically select a subset } c \text{ of } k \text{ nodes of } G.
2. \text{Test whether } G \text{ contains all edges connecting nodes in } c.
3. \text{If yes, accept; otherwise, reject."} \]

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

**How to prove a language is in NP:**
Proof technique #2: create an NTM

Don’t forget to count the steps

THEOREM
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 = \{x₁, ..., xₖ\}, and for some
  \{y₁, ..., yₙ\} ⊆ \{x₁, ..., xₖ\}, we have \(\sum yᵢ = t\)\}
  - Some subset of a set of numbers S must sum to some total t
  - e.g., \{\[4, 11, 16, 21, 27\], 25\} ∈ **SUBSET-SUM**
Theorem: \textit{SUBSET-SUM} is in NP

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

\textbf{Proof Idea} The subset is the certificate.

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

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

\( V = \text{"On input } \langle \langle S, t \rangle, c \rangle: \)
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.} \)
Proof 2: \textit{SUBSET-SUM} is in NP

\textit{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 } \Sigma y_i = t \}

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

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

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

• A composite number is **not** prime

• \text{COMPOSITES} is polynomially verifiable
  • i.e., it’s in \textbf{NP}
  • i.e., factorability is in \textbf{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)!
HW Question: Does P = NP?

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
- Optimal resource allocation could fix all the world’s (food, energy, space ...) problems?
Progress on whether $\mathbf{P} = \mathbf{NP}$?

• Some, but still not close

The Status of the $\mathbf{P}$ Versus $\mathbf{NP}$ 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:
  • $\mathbf{NP}$-Completeness
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 $B$.

**THEOREM**
If $B$ is NP-complete and $B \in \mathbb{P}$, then $\mathbb{P} = \mathbb{NP}$.

• How does this help the $\mathbb{P} = \mathbb{NP}$ problem?
Flashback: Mapping Reducibility

Language $A$ is \textit{mapping reducible} to language $B$, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$ 

The function $f$ is called the \textit{reduction} from $A$ to $B$.

**IMPORTANT:** “if and only if” ...

To show \textit{mapping reducibility}:
1. create \textit{computable fn}
2. and then show \textit{forward direction}
3. and \textit{reverse direction}
   (or \textit{contrapositive of forward direction})

$A_{TM} = \{ \langle M, w \rangle \mid M$ is a TM and $M$ accepts $w \}$

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

$A \leq_m B$ means $\overline{A} \leq_m \overline{B}$

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a \textit{computable function} if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.
Polyomial Time Mapping Reducibility

Language $A$ is *mapping reducible* to language $B$, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$ 

The function $f$ is called the *reduction* from $A$ to $B$.

Language $A$ is *polynomial time mapping reducible*, or simply *polynomial time reducible*, to language $B$, written $A \leq_P B$, if a polynomial time computable function $f: \Sigma^* \rightarrow \Sigma^*$ exists, where for every $w$,

$$w \in A \iff f(w) \in B.$$ 

The function $f$ is called the *polynomial time reduction* of $A$ to $B$.

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a *computable function* if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.
Flashback: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

**Proof** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$N = \text{“On input } w:\n1. \text{ Compute } f(w).
2. \text{ Run } M \text{ on input } f(w) \text{ and output whatever } M \text{ outputs.”}”

This proof only works because of the if-and-only-if requirement.

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

\[ w \in A \iff f(w) \in B. \]

The function $f$ is called the **reduction** from $A$ to $B$. 
Thm: If $A \leq_m B$ and $B \in \mathbb{P}$ is decidable, then $A \in \mathbb{P}$ is decidable.

**Proof** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$N = “$On input $w$:

1. Compute $f(w)$.
2. Run $M$ on input $f(w)$ and output whatever $M$ outputs.”$”

Language $A$ is *mapping reducible* to language $B$, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$ 

The function $f$ is called the *reduction* from $A$ to $B$.
Thm: If $A \leq_m B$ and $B \in \mathcal{P}$ is decidable, then $A \in \mathcal{P}$ is decidable.

**Proof** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$N = \text{"On input } w:\$
1. Compute $f(w)$.
2. Run $M$ on input $f(w)$ and output whatever $M$ outputs.”

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

\[ w \in A \iff f(w) \in B. \]

The function $f$ is called the **reduction** from $A$ to $B$.
Next Time: 3SAT is polynomial time reducible to CLIQUE.
Check-in Quiz 4/25

On gradescope