HW questions?
Announcements

• HW10 released
  • Note extended due date: Sun Dec 6 11:59pm EST

• HW7 and 8 resubmissions due Mon Nov 30 11:59pm EST
Recap: The *PATH* Problem

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

• The **search** problem:
  • Exponential time (brute force) algorithm:
    • 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
Verifying a *PATH*

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

- **The verification problem:**
  - Given some path \( p \) in \( G \), check that it is a path from \( s \) to \( t \)
  - Let \( m = \) longest possible path = \# edges in \( G \)

- **Verifier \( V = \) On input \( <G, s, t, p> \), where \( p \) 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. 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
     - **Max steps** of each loop iteration \( O(m) \)
     - **Loop iterates** 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** = \( O(m) + O(m^2) = O(m^2) = \) polynomial in \( m \)

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

PATH = \{⟨G, s, t⟩| G is a directed graph that has a directed path from s to t\}

**Definition 7.18**

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

\[ A = \{w | V \text{ accepts } ⟨w, c⟩ \text{ for some string } c\}. \]

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?
• \(PATH\) is polynomially verifiable
The **HAMPATH** Problem

- A Hamiltonian path goes through every node in the graph

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

- 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 7.19**

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

- *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.

• \( \Rightarrow \) If a lang \( L \) is in NP, then it has a poly time verifier \( V \)
• Create NTM deciding \( L \): on input \( w = \)
  • Nondeterministically run \( V \) with \( w \) and all possible certs \( c \)
• \( \Leftarrow \) If \( L \) has NTM decider \( N \),
  • then let the cert denote one accepting path in \( N \)
  • Then create poly time verifier that runs \( N \) for only that path
**P vs NP**

**Definition 7.7**
Let $t: \mathbb{N} \to \mathbb{R}^+$ 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**
$\text{NTIME}(t(n)) = \{L | L$ is a language decided by an $O(t(n))$ time nondeterministic Turing machine$\}.$

**Corollary 7.22**

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

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

- $\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}$
  - Some subset of a set of numbers sums to some total
  - e.g., $\langle \{4, 11, 16, 21, 27\}, 25 \rangle \in \text{SUBSET-SUM}$
Theorem: **CLIQUE is in NP**

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

**PROOF IDEA** The clique is the certificate.

**PROOF** The following is a verifier $V$ for $CLIQUE$.

$V =$ "On input $\langle G, k \rangle, c$:

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

$A = \{ w | V \text{ accepts } \langle w, c \rangle \text{ for some string } c \}$.

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**

NP is the class of languages that have polynomial time verifiers.
Proof 2: \textbf{CLIQUE} is in NP

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

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

\[ \text{“try all subgraphs”} \]

\[ O(k^2) \]

\textbf{THEOREM 7.20}

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

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

\( V = \) “On input \( \langle S, t \rangle, c \):

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 non-deterministic 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
  - 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)
Does $P = NP$?

One of the greatest unsolved mysteries in math and computer science

**PATH**

**NP**

**P**

**CLIQUE**

**HAMPATH**

**COMPOSITES**

**PROOF:**

$e^{P_i} = 1$

and $P_i \leq P_i$

$e^{P_i} = e^{P_i - i \cdot e^p}$

$e^{P_i - i} = e^{P_i}$

which leaves

$P = 0$

Thus $P = NP$

QED

It's hard to prove that something doesn't exist
Check-in Quiz 11/25
On gradescope

End of Class Survey 11/25
See course website