CSE 135: Introduction to Theory of Computation
P, NP, and NP-completeness

Sungjin Im

University of California, Merced

04-23-2015

Significance of $P \neq_? NP$

Perhaps you have heard of (some of) the following terms:
P, NP, NP-complete, NP-hard.

And the famous question if $P = NP$ or not.

Significance of $P \neq_? NP$

Why do we care about $P \neq_? NP$?

Significance of $P \neq_? NP$

Why do we care about $P \neq_? NP$?

- A central question in computer science and mathematics.

Significance of $P \neq_? NP$

Why do we care about $P \neq_? NP$?

- A central question in computer science and mathematics.
- A lot of practical implications.

Significance of $P \neq_? NP$

Why do we care about $P \neq_? NP$?

- A central question in computer science and mathematics.
- A lot of practical implications.
- P and NP problems are widely found in practice.

Significance of $P \neq_? NP$

The Millennium Prize Problems are seven problems in mathematics that were stated by the Clay Mathematics Institute in 2000. As of April 2015, six of the problems remain unsolved. A correct solution to any of the problems results in a US $1,000,000 prize (sometimes called a Millennium Prize) being awarded by the institute. The Poincare conjecture was solved by Grigori Perelman, but he declined the award in 2010.

The question $P \neq_? NP$ is the first in the list...

(source: wikipedia)

Class P

Formal definition

Definition

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

$$P = \bigcup_k Time(n^k)$$

Motivation: To define a class of problems that can be solved efficiently.

- P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing Machine.
- P roughly corresponds to the class of problems that are realistically solvable on a computer.

Class P

Justification

P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing Machine.

Example.

Theorem

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

In fact, it can be shown that all reasonable deterministic computational models are polynomially equivalent.

Class P

Formal definition

Definition

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

$$P = \bigcup_k Time(n^k)$$

Hence, a language is in P if and only if one can write a pseudo-code that decides the language in polynomial time in the input length; the code must terminate for any input.

Class P

Example

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

Class P

Less formal definition of P

Definition

P is the set of decision problems that can be solved in polynomial time (in the input size).

Examples of polynomial time: $O(n)$, $O(\log n)$, $O(n^{100})$, $O(n^{2^{2^{2^2}}})$.

Class P

Less formal definition of P

Definition

P is the set of decision problems that can be solved in polynomial time (in the input size).

Examples of polynomial time: $O(n)$, $O(\log n)$, $O(n^{100})$, $O(n^{2^{2^{2^2}}})$.

From now on, we will use this less formal and simpler definition of P.

Simple description of decision problems

Problem $PATH$:
Input: an directed graph G, and two distinct nodes s and t in G.
Question: Does G has a directed path from s to t?

In the remainder of this course, we will adopt this simple description of decision problems over languages.

Simple description of decision problems

Problem *PATH*:
Input: an directed graph G, and two distinct nodes s and t in G.
Question: Does G has a directed path from s to t?

In the remainder of this course, we will adopt this simple description of decision problems over languages.

A yes-instance is an instance where the answer is yes. No-instance is similarly defined.

Class *NP*

Example

Problem *HAMPATH*:
Input: an directed graph G, and two distinct nodes s and t in G.
Question: Does G have a Hamiltonian path from s to t?

A Hamiltonian path in a directed graph G is a directed path that visits every node exactly once.

Class *NP*

Example

Problem *HAMPATH*:
Input: an directed graph G, and two distinct nodes s and t in G.
Question: Does G have a Hamiltonian path from s to t?

A Hamiltonian path in a directed graph G is a directed path that visits every node exactly once.

We are not aware of any algorithm that solves *HAMPATH* in polynomial time. But we know a brute-force algorithm finds a s-t Hamiltonian path in exponential time. Also we can verify/check if a given path is a s-t Hamiltonian path or not.

Class *NP*

Example

We do not know how to answer in polynomial time if a given instance is a yes-instance or not. However, if it is a yes-instance, there is a proof I can easily check (in polynomial time). A s-t Hamiltonian path of the instance can be such a 'proof'. Once we are given this proof, we can check in polynomial time if the instance is indeed a yes-instance

Class *NP*

Formal definition

Definition

A verifier for a language A is an algorithm V, where

$$A = \{w \mid V \text{ accepts } < w, c > \text{ for some string } c\}$$

(c is called a certificate or proof).
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 that a polynomial time verifier A can only read a certificate of size polynomial in $|w|$; so c must have size polynomial in $|w|$.

Class *NP*

Formal definition

Definition

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

Class *NP*

Formal definition

Class *NP*

Formal definition

The term *NP* comes from nondeterministic polynomial time and has an alternative characterization by using nondeterministic polynomial time Turing machines.

Theorem

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

Proof.

($\Rightarrow$) Convert a polynomial time verifier V to an equivalent polynomial time NTM N. On input w of length n:

- Nondeterministically select string c of length at most n^k (assuming that V runs in time n^k).
- Run V on input $< w, c >$.
- If V accepts, accept; otherwise, reject.

Class *NP*

Formal definition

Theorem

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

Proof.

($\Leftarrow$) Convert a polynomial time NTM N to an equivalent polynomial time verifier V. On input w of length n:

- Simulate N on input w, treating each symbol of c as a description of the nondeterministic choice to make at each step.
- If this branch of N's computation accepts, accept; otherwise, reject.

□

Class *NP*

Examples

A clique in an undirected graph is a subgraph, wherein every two nodes are connected by an edge. A k-clique is a clique that contains k nodes.

$$CLIQUE = \{< G, k > \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

Lemma

CLIQUE is in NP.

Proof.

Let $w =< G, k >$. The certificate c is a k-clique. We can easily test if c is a clique in polynomial time in w and c, and has k nodes. □

Class *NP*

Examples

$$HAMPATH = \{< G, s, t > \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}.$$

Lemma

HAMPATH is in NP.

Simple definition of P

From now on, we will use the following simpler definition of P.

Definition

P is the set of decision problems that can be solved in polynomial time (in the input size).

Examples of polynomial time: $O(n)$, $O(\log n)$, $O(n^{100})$, $O(n^{2^{2^{2^2}}})$, etc.

Class *NP*

Less formal definition

From now on, we will use the following simpler definition of *NP*.

Definition

NP is the set of decision problems with the following property: If the answer is Yes, then there is a proof of this fact that can be checked in polynomial time.

(The size of the proof must be polynomially bounded by n).

Class NP

Example

Problem *CLIQUE*:
Input: an undirected graph G and an integer $k \geq 1$.
Question: Does G have a k-clique?

A clique in an undirected graph is a subgraph, wherein every two nodes are connected by an edge. A k-clique is a clique that contains k nodes.

Proposition

CLIQUE is in NP.

Proof.

The certificate is a set of k vertices that form a clique which can be checked in polynomial time. □

Class NP

Example

Problem *HAMPATH*:
Input: an directed graph G, and two distinct nodes s and t in G.
Question: Does G have a Hamiltonian path from s to t?

Proposition

HAMPATH is in NP.

Proof.

Consider any yes-instance. The certificate is the following: a Hamiltonian path from s to t which must exist from the definition of the problem, and can easily be checked in polynomial time. □

$P \subseteq NP$

Think about any decision problem A in the class P. Why is it in NP?

$P \subseteq NP$

Think about any decision problem A in the class P. Why is it in NP?

In other words, if an input/instance is a Yes-instance, how can we check it in polynomial time? We can solve the problem from scratch in polynomial time. No certificates are needed.

$P \subseteq NP$

Think about any decision problem A in the class P. Why is it in NP?

In other words, if an input/instance is a Yes-instance, how can we check it in polynomial time? We can solve the problem from scratch in polynomial time. No certificates are needed.

For example, think about the probelm $PATH$.

$NP \subseteq EXP$

Here, EXP is the class of problems that can be solved in exponential time.

$NP \subseteq EXP$

Here, EXP is the class of problems that can be solved in exponential time.

Think about any decision problem A in the class NP.

$NP \subseteq EXP$

Here, EXP is the class of problems that can be solved in exponential time.

Think about any decision problem A in the class NP. If a yes-instance has a 'short' certificate. We can try each certificate (by brute force). So the checking can be done in exponential time.

$NP \subseteq EXP$

Here, *EXP* is the class of problems that can be solved in exponential time.

Think about any decision problem *A* in the class *NP*. If a yes-instance has a 'short' certificate. We can try each certificate (by brute force). So the checking can be done in exponential time.

For example, think about *CLIQUE* or *HAMPATH*.

P vs. NP?

Either of the following two must be true:
$P = NP$ or $P \subsetneq NP$.

We know that $NP \subseteq EXP$, but we do not even know if $NP = EXP$ or $NP \subsetneq EXP$

Most researchers, however, believe that $P \neq NP$ becuase of the existence of some problems that capture the *entire* NP class.

Problem Satisfiability (SAT):
Input: a boolean formula Φ.
Question: is Φ satisfiable?

A Boolean formula is an expression involving Boolean variables and operations. For example, $\Phi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$ is a boolean formula. We say a boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1. Note that Φ is satisfiable $(x = 0, y = 1, z = 0)$.

Theorem (Cook-Levin Theorem)

$SAT \in P$ iff $P = NP$.

There are two different types of reductions: Turing reduction vs. Karp reduction. For simplicity, in this course we will only focus on Turing reduction. See Section 30.4 in Jeff Erickson's note for the difference.

Definition

We say that problem A is polynomial-time reducible to problem B if we can solve problem A in polynomial time using a (possibly hypothetical) polynomial time algorithm for problem B as a black box. This is often denoted as $A \leq_p B$.

For simplicity, sometimes, we say that A is reducible to B, and denote it as $A \leq B$.

Definition

We say that problem B is NP-hard if for **every** A in NP, $A \leq_P B$.

NP-completeness

NP-hardness

Theorem

If $A \leq_P B$ and $B \in P$, then $A \in P$.

Theorem

If $A \leq_P B$ and $B \in P$, then $A \in P$.

Proof.

Immediately follows from the definition of $\leq_p$. □

Definition

A language B is NP-complete if it is NP-hard and is in NP.

Definition

A language B is NP-complete if it is NP-hard and is in NP.

Theorem

If B is NPC (NP-complete) and $B \in P$, then $P = NP$.

Proof.

We already know $P \subseteq NP$. So it suffices to show $NP \subseteq P$.

Definition

A language B is NP-complete if it is NP-hard and is in NP.

Theorem

If B is NPC (NP-complete) and $B \in P$, then $P = NP$.

Proof.

We already know $P \subseteq NP$. So it suffices to show $NP \subseteq P$. Consider any problem A in NP. By definition of NP-hardness, $A \leq_p B$. Since B is in P, it implies A is in P. □

NP-completeness

Let's say that $B \in NPC$ if B is NP-complete.

Theorem

A language C is NP-complete if $B \leq_P C$ for some $B \in NPC$, and $C \in NP$.

Similarly,

Theorem

A language C is NP-hard if $B \leq_P C$ for some NP-hard language B.

Theorem

SAT is NP-complete.

NP-completeness

NPC derivation tree

Theorem

SAT is NP-complete.