Introduction to P, NP, & NP-Completeness

The $1M question

The Clay Mathematics Institute
Millenium Prize Problems

1. Birch and Swinnerton-Dyer Conjecture
2. Hodge Conjecture
3. Navier-Stokes Equations
4. P vs NP
5. Poincaré Conjecture
6. Riemann Hypothesis
7. Yang-Mills Theory

The P versus NP problem

Is perhaps one of the biggest open problems in computer science (and mathematics!) today.

(Even featured in the TV show NUMB3RS)

But what is the P-NP problem?

Sudoku

2			3		8		5	
		3		4	5	9	8	
		8			9	7	3	4
6		7		9				
9	8						1	7
				5		6		9
3	1	9	7			2		
	4	6	5	2		8		
	2		9		3			1

3x3x3

Sudoku

2	9	4	3	7	8	1	5	6
1	7	3	6	4	5	9	8	2
5	6	8	2	1	9	7	3	4
6	5	7	1	9	2	3	4	8
9	8	2	4	3	6	5	1	7
4	3	1	8	5	7	6	2	9
3	1	9	7	8	4	2	6	5
7	4	6	5	2	1	8	9	3
8	2	5	9	6	3	4	7	1

3x3x3

Sudoku

	F		2						6			C	B	3	
	C				4	8	E	A			0		D		
D	A	8			3		2	7	F			6		5	
6			E	D	F		C		8						7
	9	3		7					A						2
E						6	F	5		8	4		3		1
C	8		1	3	9	D		0	2		E				
	D		6		5	E	B		1					0	4
9	6					1		F	3	2		0		A	
				4		A	8		D	0	9	B		2	5
2		A		0	D		5	6	C						F
5						2					A		4	8	
B						4		1		A	2	F			0
	0		7			F	3	C		D			2	9	B
		5		1			A	9	0	B				D	
	2	D	A			9						1		4	

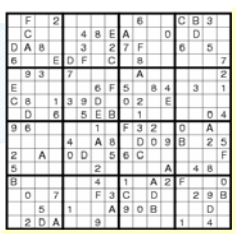
4x4x4

Sudoku

0	F	9	2	A	7	5	1	4	6	E	D	C	B	3	8
7	C	1	3	6	4	8	E	A	B	5	0	2	D	F	9
D	A	8	4	9	3	B	2	7	F	C	1	6	0	5	E
6	5	B	E	D	F	0	C	2	8	9	3	4	A	1	7
4	9	3	5	7	1	C	0	D	A	F	B	8	E	6	2
E	B	7	0	2	A	6	F	5	9	8	4	D	3	C	1
C	8	F	1	3	9	D	4	0	2	6	E	5	7	B	A
A	D	2	6	8	5	E	B	3	1	7	C	9	F	0	4
9	6	4	8	E	B	1	7	F	3	2	5	0	C	A	D
3	7	C	F	4	6	A	8	E	D	0	9	B	1	2	5
2	1	A	B	0	D	3	5	6	C	4	8	7	9	E	F
5	E	0	D	F	C	2	9	B	7	1	A	3	4	8	6
B	3	6	9	C	E	4	D	1	5	A	2	F	8	7	0
1	0	E	7	5	8	F	3	C	4	D	6	A	2	9	B
8	4	5	C	1	2	7	A	9	0	B	F	E	6	D	3
F	2	D	A	B	0	9	6	8	E	3	7	1	5	4	C

4x4x4

Sudoku



n x n x n

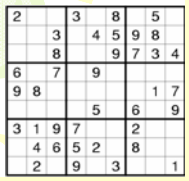
Suppose it takes you S(n) to solve n x n x n

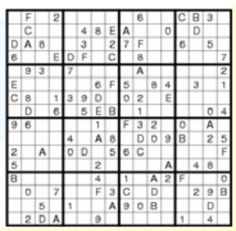
V(n) time to verify the solution

Fact: $V(n) = O(n^2 \times n^2)$

Question: is there some constant such that

$S(n) \leq [V(n)]^{constant}$?





⋮

n x n x n

Sudoku

P vs NP problem

=

Does there exist an algorithm for n x n x n sudoku that runs in time P(n) for some polynomial P(n)?

The P versus NP problem (informally)

Is proving a theorem much more difficult than checking the proof of a theorem?

X problems?

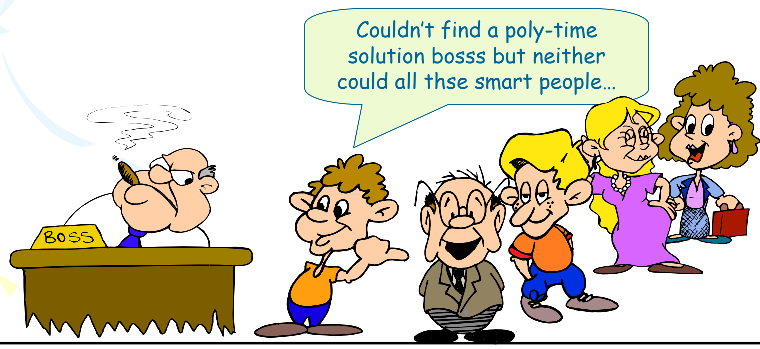
- X = Hard ? Tough? Herculean? Formidable? Arduous? NPC?
- X = Impractical? Bad? Heavy? Tricky? Intricate? Prodigious ? Difficult? Intractable? Costly ? Obdurate? Obstinate ? Exorbitant? Interminable?

X problems?

- X = Hard ? Tough? Herculean? Formidable? Arduous?
- X = Impractical? Bad? Heavy? Tricky? Intricate? Prodigious ? Difficult? Intractable? Costly ? Obdurate? Obstinate ? Exorbitant? Interminable?

X problems?

- X = Hard ? Tough? Herculean? Formidable? Arduous?
- X = Impractical? Bad? Heavy? Tricky? Intricate? Prodigious ? Difficult? Intractable? Costly ? Obdurate? Obstinate ? Exorbitant? Interminable?



Let's start at the beginning…

Polynomial Problems P

- Are solvable in polynomial time
- Are solvable in $O(n^k)$, where k is some constant.
- Most of the algorithms we have covered are in P

Nondeterministic Polynomial (NP) Problems

- This class of problems has solutions that are verifiable in polynomial time.
  - Thus any problem in P is also NP, since we would be able to solve it in polynomial time, we can also verify it in polynomial time
- NP does not stand for nonpolynomial

NP-Complete Problems

- Is an NP-Problem
- Is at least as difficult as an NP problem (is reducible to it)

- More formally, a decision problem C is NP-Complete if:
  - C is in NP
  - Any known NP-hard (or complete) problem $\leq_p$ C
  - Thus a proof must show these two being satisfied

Reduction

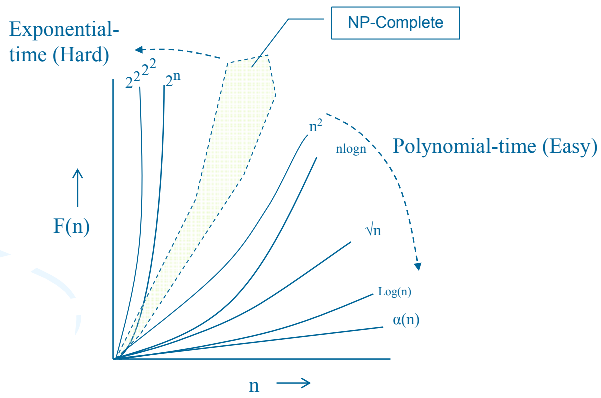
- P1 : is an unknown problem (easy/hard ?)
- P2 : is known to be difficult

If we can easily solve P2 using P1 as a subroutine then P1 is difficult

Must create the inputs for P1 in polynomial time.

* P1 is definitely difficult because you know you cannot solve P2 in polynomial time unless you use a component that is also difficult (it cannot be the mapping since the mapping is known to be polynomial)

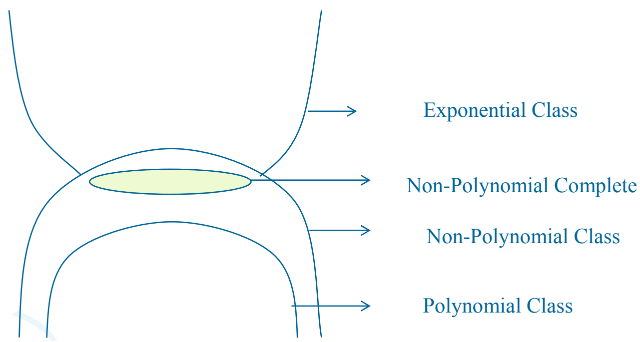
Exponential Time Algorithms



Examples

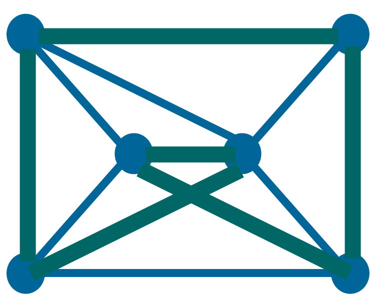
- Longest path problem: (similar to Shortest path problem, which requires polynomial time) suspected to require exponential time, since there is no known polynomial algorithm.
- Hamiltonian Cycle problem: Traverses all vertices exactly once and form a cycle.

Where does NP Complete lie?



Hamilton Cycle

Given a graph G = (V,E), a cycle that visits all the nodes exactly once



The Problem “HAM”

Input: Graph G = (V,E)

Output: YES if G has a Hamilton cycle

NO if G has no Hamilton cycle

The Set “HAM”

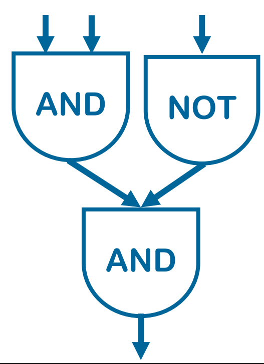
HAM = { graph G | G has a Hamilton cycle }

Circuit-Satisfiability

Input: A circuit C with one output

Output: YES if C is satisfiable

NO if C is not satisfiable



The Set “SAT”

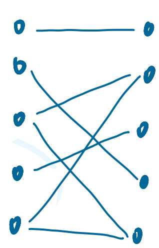
SAT = { all satisfiable circuits C }

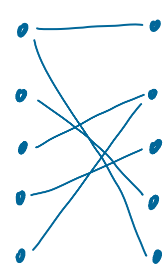
Bipartite Matching

Input: A bipartite graph G = (U,V,E)

Output: YES if G has a perfect matching

NO if G does not





The Set “BI-MATCH”

BI-MATCH = { all bipartite graphs that have a perfect matching }

Sudoku

Input: n x n x n sudoku instance

Output: YES if this sudoku has a solution

NO if it does not

The Set “SUDOKU”

SUDOKU = { All solvable sudoku instances }

Decision Versus Search Problems

Decision Problem

YES/NO
Does G have a Hamilton cycle?

Search Problem

Find a Hamilton cycle in G if one exists, else return NO

Reducing Search to Decision

Given an algorithm for decision Sudoku, devise an algorithm to find a solution

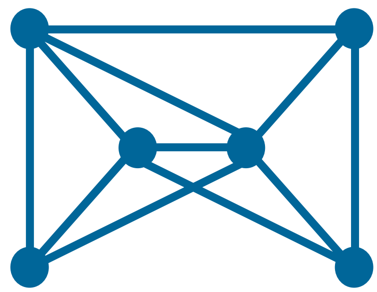
Idea:
Fill in one-by-one and use decision algorithm

2			3		8		5	
		3		4	5	9	8	
		8			9	7	3	4
6		7		9				
9	8						1	7
				5		6		9
3	1	9	7			2		
	4	6	5	2		8		
	2		9		3			1

Reducing Search to Decision

Given an algorithm for decision HAM, devise an algorithm to find a solution

Idea:
Find the edges of the cycle one by one



Decision/Search Problems

We'll study decision problems because they are almost the same (asymptotically) as their search counterparts

Polynomial Time and The Class “P” of Decision Problems

What is an efficient algorithm?

Is an O(n) algorithm efficient?

How about O(n log n)?

$O(n^2)$?

$O(n^{10})$?

polynomial time

$O(n^c)$ for some constant c

$O(n^{\log n})$?

$O(2^n)$?

$O(n!)$?

non-polynomial time

Does an algorithm running in $O(n^{100})$ time count as efficient?

We consider non-polynomial time algorithms to be inefficient.

And hence a necessary condition for an algorithm to be efficient is that it should run in poly-time.

Asking for a poly-time algorithm for a problem sets a (very) low bar when asking for efficient algorithms.

The question is: can we achieve even this?

The Class P

We say a set $L \subseteq \Sigma^*$ is in **P** if there is

a program A and
a polynomial p()

such that for any x in Σ^*,

A(x) runs for at most p(|x|) time
and answers question "is x in L?" correctly.

The Class P

The class of all sets L that can be recognized in polynomial time.

The class of all decision problems that can be decided in polynomial time.

Why are we looking only at sets $\subseteq \Sigma^*$?

What if we want to work with graphs or boolean formulas?

Languages/Functions in P?

Example 1:
CONN = {graph G: G is a connected graph}

Algorithm A_1:
If G has n nodes, then run depth first search from any node, and count number of distinct node you see. If you see n nodes, G $\in$ CONN, else not.

Time: $p_1(|x|) = \Theta(|x|)$.

Languages/Functions in P?

HAM, SUDOKU, SAT are not known to be in P

CO-HAM = { G | G does NOT have a Hamilton cycle}

CO-HAM $\in$ P if and only if HAM $\in$ P

Onto the new class, NP

Verifying Membership

Is there a short “proof” I can give you for:

G $\in$ HAM?

G $\in$ BI-MATCH?

G $\in$ SAT?

G $\in$ CO-HAM?

NP

A set $L \in NP$

if there exists an algorithm A and a polynomial p( )

For all $x \in L$	For all $x' \notin L$
there exists y with $\|y\| \le p(\|x\|)$	For all y′ with $\|y'\| \le p(\|x'\|)$
such that A(x,y) = YES	we have A(x′,y′) = NO
in p(|x|) time	in p(|x|) time

Recall the Class P

We say a set $L \subseteq \Sigma^*$ is in **P** if there is

a program A and
a polynomial p()

such that for any x in Σ^*,

A(x) runs for at most p(|x|) time
and answers question "is x in L?" correctly.

can think of A as "proving" that x is in L

NP

A set $L \in NP$

if there exists an algorithm A and a polynomial p( )

For all $x \in L$	For all $x' \notin L$
there exists a y with $\|y\| \le p(\|x\|)$	For all y' with $\|y'\| \le p(\|x'\|)$
such that A(x,y) = YES	Such that $A(x',y')$ = NO
in $p(\|x\|)$ time	in $p(\|x\|)$ time

The Class NP

The class of sets L for which there exist "short" proofs of membership
(of polynomial length)
that can "quickly" verified
(in polynomial time).

Recall: A doesn't have to find these proofs y; it just needs to be able to verify that y is a "correct" proof.

$P \subseteq NP$

For any L in P, we can just take y to be the empty string and satisfy the requirements.

Hence, every language in P is also in NP.

Languages/Functions in NP?

$G \in$ HAM?

$G \in$ BI-MATCH?

$G \in$ SAT?

$G \in$ CO-HAM?

Summary: P versus NP

Set L is in P if membership in L can be decided in poly-time.

Set L is in NP if each x in L has a short "proof of membership" that can be verified in poly-time.

Fact: $P \subseteq NP$

Question: Does $NP \subseteq P$?

Why Care?

NP Contains Lots of Problems We Don’t Know to be in P

- Classroom Scheduling
- Packing objects into bins
- Scheduling jobs on machines
- Finding cheap tours visiting a subset of cities
- Allocating variables to registers
- Finding good packet routings in networks
- Decryption
- …

OK, OK, I care. But Where Do I Begin?

How can we prove that
$NP \subseteq P$?

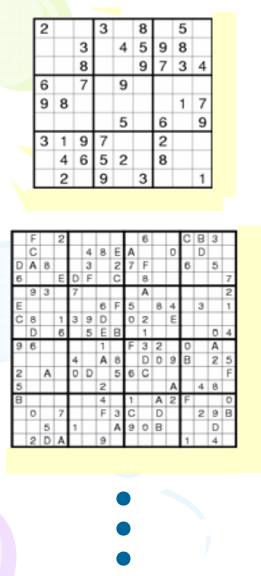
I would have to show that
every set in NP has a
polynomial time algorithm…

How do I do that?
It may take forever!
Also, what if I forgot one of
the sets in NP?

We can describe
one problem L in NP,
such that
if this problem L is in P,
then $NP \subseteq P$.

It is a problem that can
capture all other problems
in NP.

The “Hardest” Set in NP



n x n x n

Sudoku

Sudoku has a polynomial time algorithm if and only if P = NP

The “Hardest” Sets in NP

Sudoku

Clique

SAT

Independent-Set

3-Colorability

HAM

How do you prove these are the hardest?

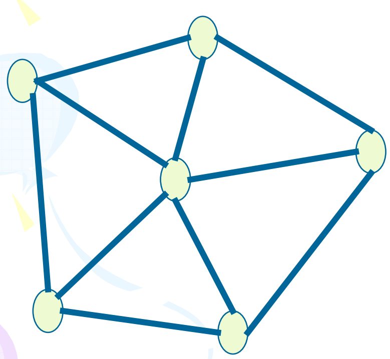
Theorem [Cook/Levin]:
SAT is one language in NP, such that if we can show SAT is in P, then we have shown $NP \subseteq P$.

SAT is a language in NP that can capture all other languages in NP.

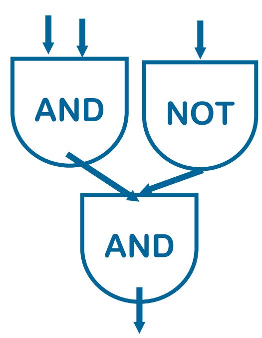
We say SAT is NP-complete.

Last lecture...

3-colorability



Circuit Satisfiability



Last lecture...

SAT and 3COLOR: Two problems that seem quite different, but are substantially the same.

Also substantially the same as CLIQUE and INDEPENDENT SET.

If you get a polynomial-time algorithm for one,
you can get a polynomial-time algorithm for ALL.