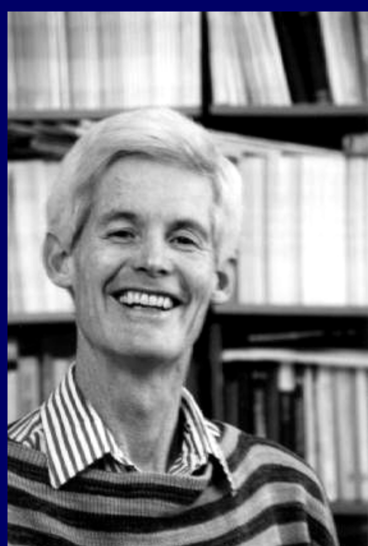
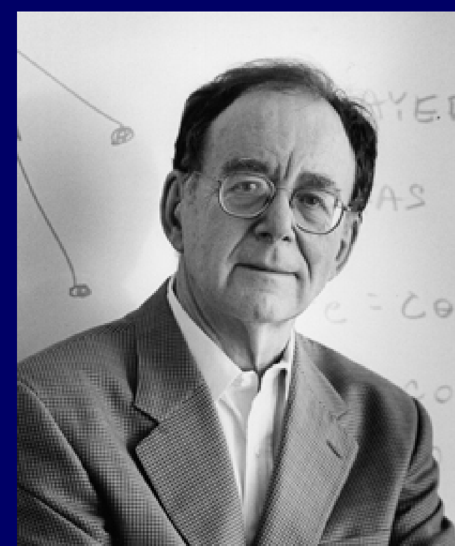
Great Theoretical Ideas In Computer Science

Anupam Gupta CS 15-251 Fall 2006

Lecture 28 Dec 7th, 2006 Carnegie Mellon University

Complexity Theory: the P vs NP question





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

http://www.claymath.org/millennium/

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

3×3×3

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	

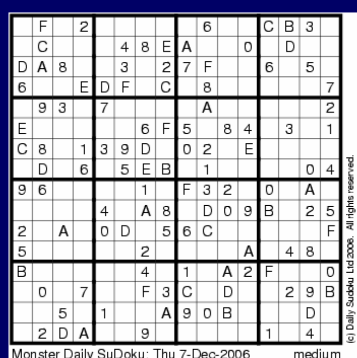
4x4x4

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



⋮

$n \times n \times n$

Suppose it takes you

$S(n)$ time to solve $n \times n \times n$

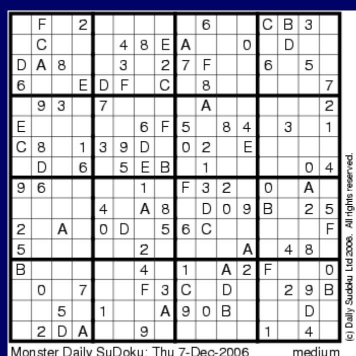
$V(n)$ to verify the solution

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

Q: is there some constant such

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

SUDOKU



⋮

$n \times n \times n$

P-vs-NP problem

=

Does there exist an algorithm for $n \times n \times n$ Sudoku that runs in time $p(n)$ for some polynomial $p(\cdot)$?

The P versus NP problem (informally)

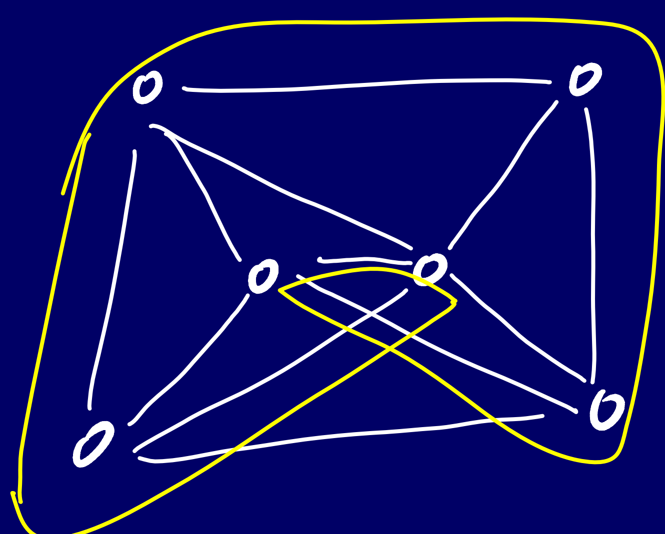
Is proving a theorem
<u>much</u> more difficult than
checking the proof of a theorem?

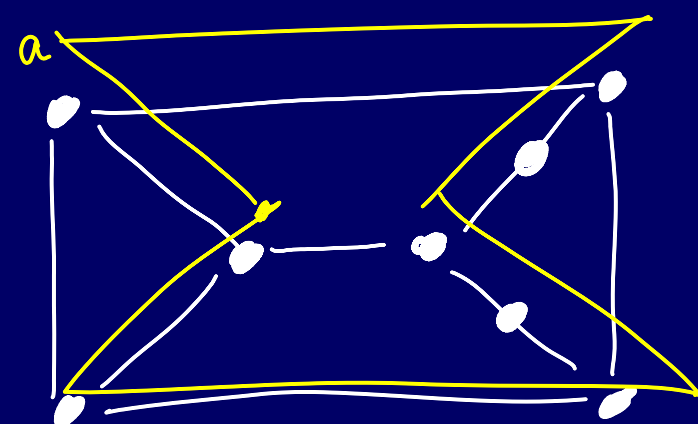
Loosely, that is what the P-vs-NP question asks.

Let's start at the beginning…

HAMILTON ~~PATH~~ CYCLE

Given a graph $G=(V,E)$, a path that visits all the nodes without visiting any node twice.





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"

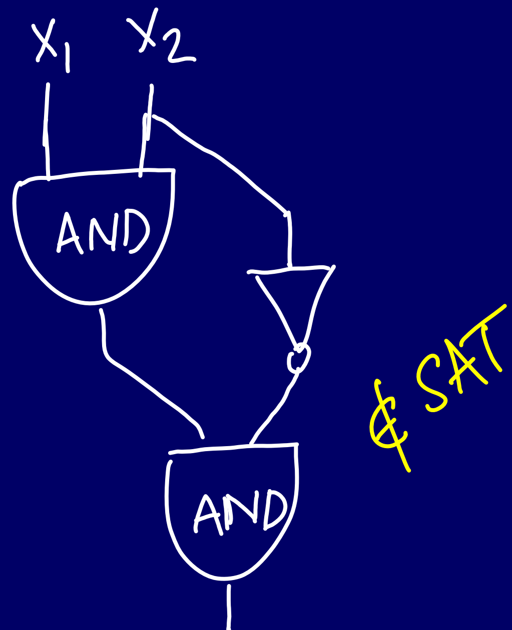
HAM = { graphs G | G has a hamilton cycle }

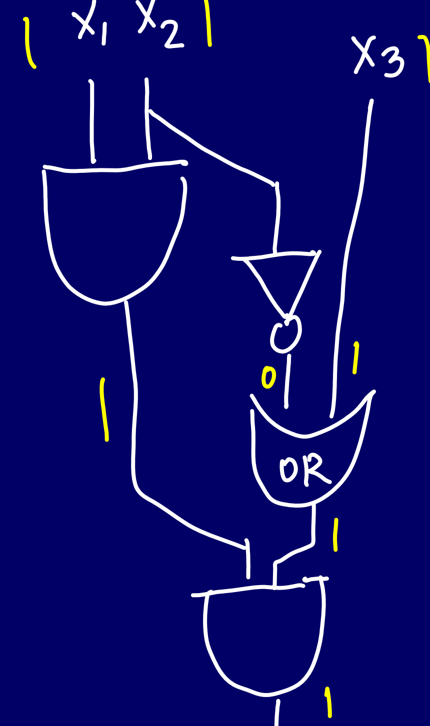
CIRCUIT SAT

Input: a combinatorial circuit C with one output

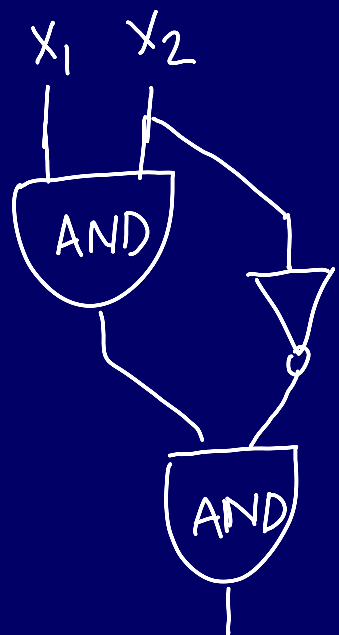
Output: YES, if C is satisfiable

NO, if C is not satisfiable

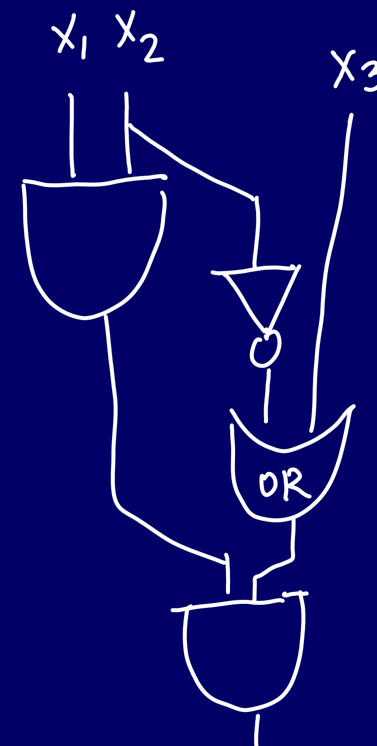




Circuit Sat



$(x_1 \land x_2) \land (\neg x_2)$



$(x_1 \land x_2) \land (\neg x_2 \lor x_3)$

THE SET "SAT"

SAT = { all satisfiable circuits C }

any $C \in SAT$

has <u>at least</u> one satisfying assigment

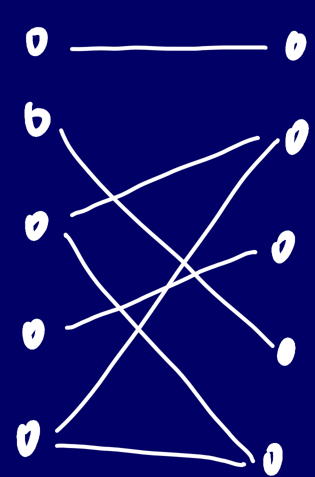
of T/F to its variables.

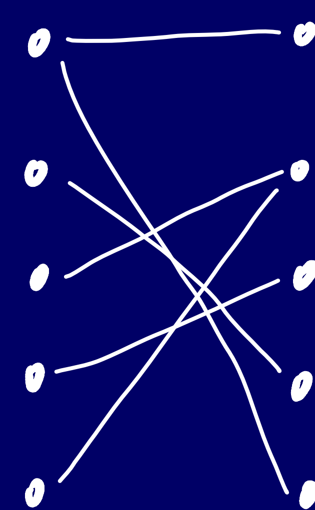
BIPARTITE MATCHING

<u>Input</u>: a bipartite graph $G = (U, V, E)$

<u>Output</u>: YES if G has a perfect matching

NO if G has <u>no</u> perfect matching





THE SET "BI-MATCH"

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

SUDOKU

Input: $n \times n \times n$ Sudoku instance

Output. YES : this Sudoku has a solution

NO : has no solution

THE SET "SUDOKU"

SUDOKU = { all solvable Sudoku instances }

DECISION VS SEARCH PROBLEMS

Decision Problem	Search Problem
YES/NO	
Does G have a Hamilton cycle?	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

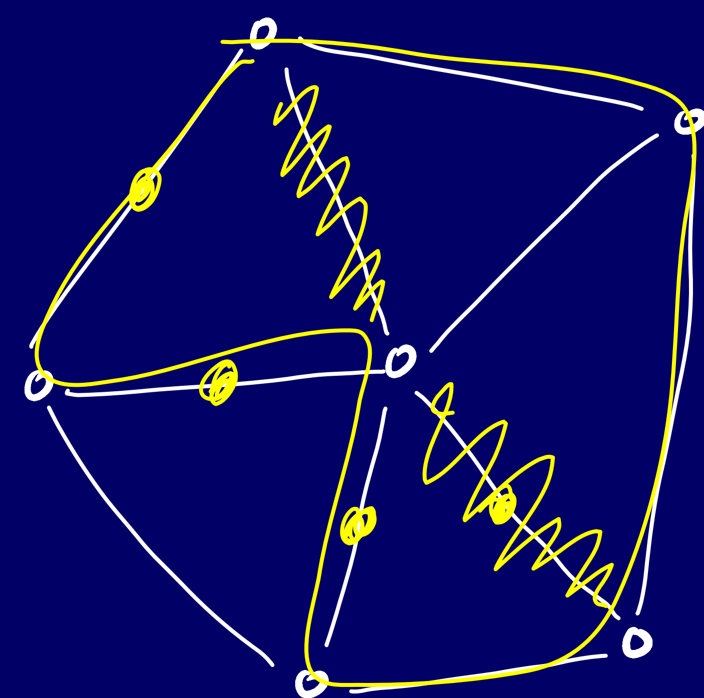
2			3		8		5	
		3		4	5	9	8	
		8			9	7	3	4
6		7		9				
9	8						1	7
				5		6		9
3	1	9	7			2		
	4	6	5	2		8		
	2		9		3			1

REDUCING SEARCH TO DECISION

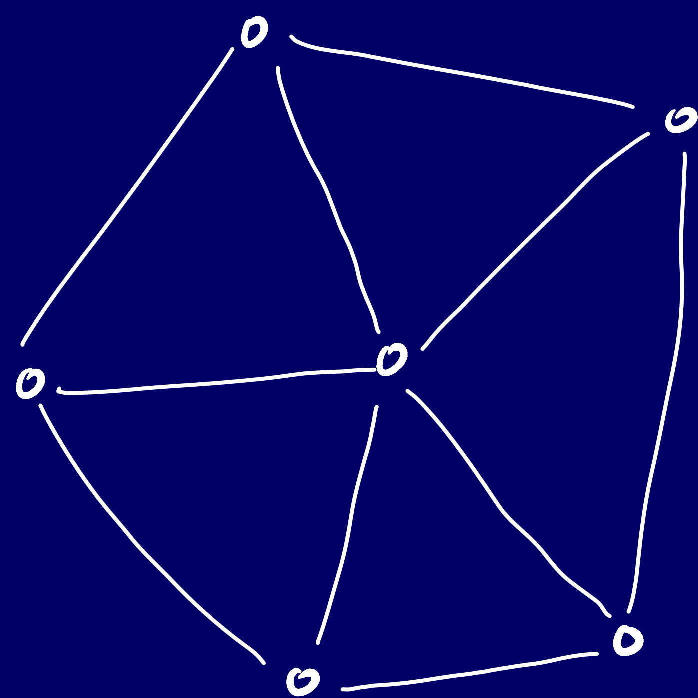
Given an algorithm for decision HAM, devise an algorithm to find the cycle

Idea:

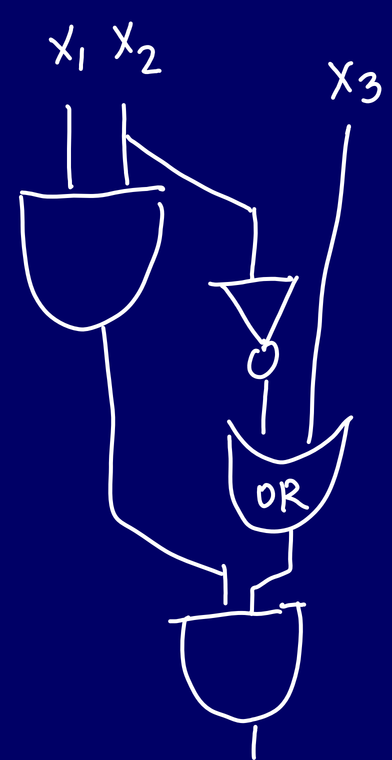
One by one, find the edges in the cycle.



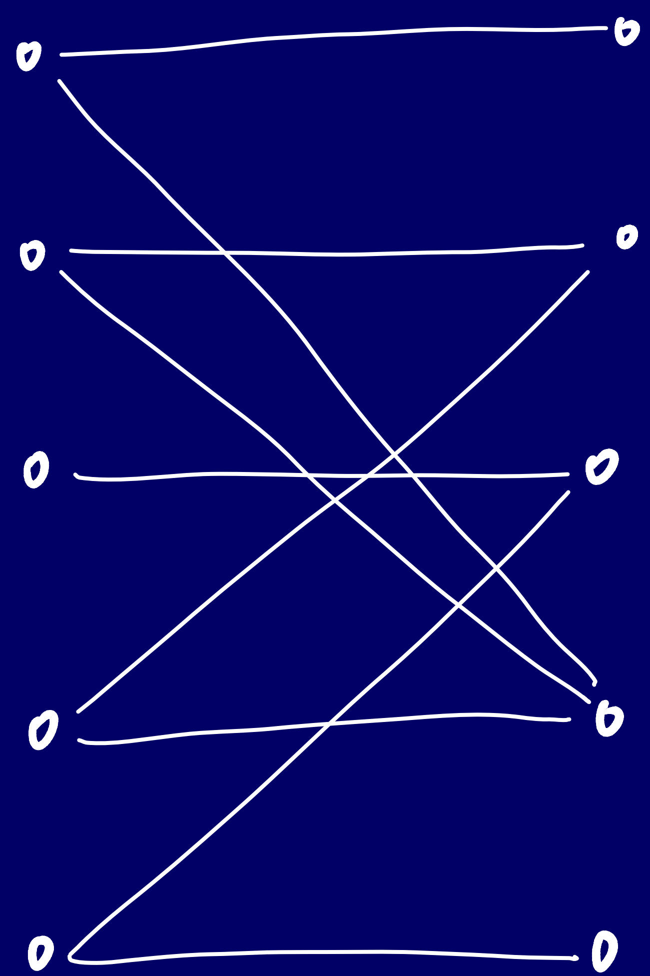
REDUCING SEARCH TO DECISION



REDUCING SEARCH TO DECISION



Reducing Search to Decision



SEARCH / DECISION PROBLEMS

We'll study decision problems
(almost the same as their search counterparts)

HAM

SAT

Bi-MATCH.

languages

View both as <u>problems</u> and <u>sets</u>

POLYNOMIAL TIME

AND

THE CLASS 'P' of DECISION PROBLEMS

What is an efficient algorithm?

Is an $O(n)$ time algorithm efficient?

How about $O(n \log n)$?

$O(n^2)$?

$O(n^{10})$?

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

$O(2^n)$?

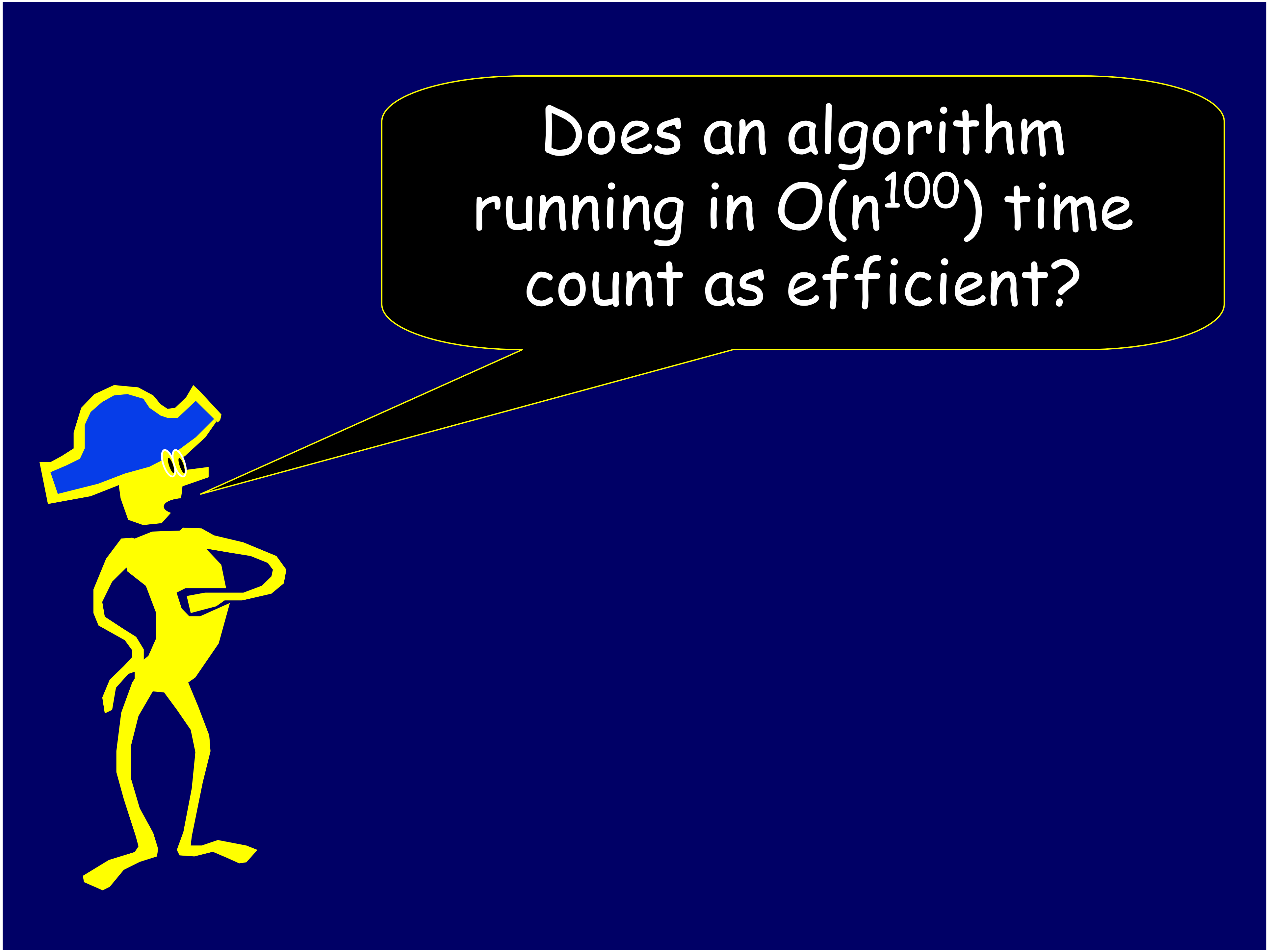
$O(n!)$?

$O(2^{2^{2n}})$?

polynomial time

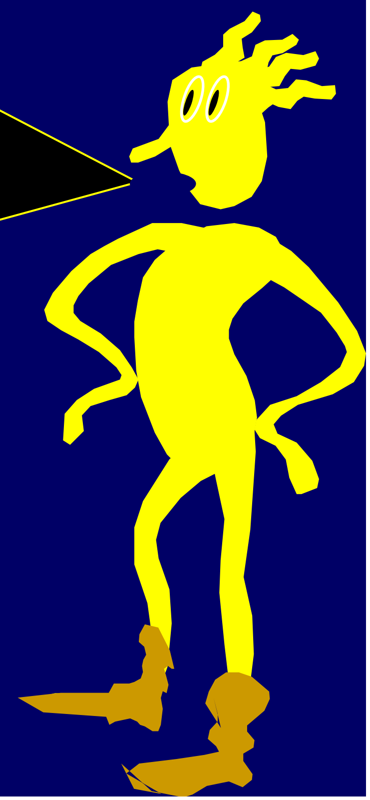
$O(n^c)$ for some constant c

non-polynomial time


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

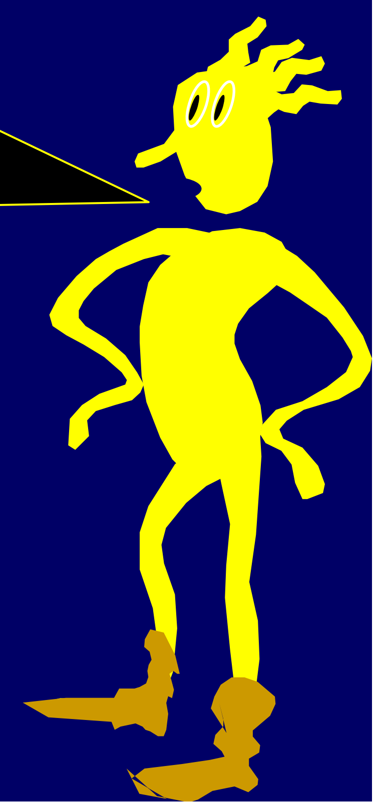
We consider <u>non-polynomial</u> time algorithms to be <u>inefficient</u>.

And hence a <u>necessary</u> 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?



I see!

Once we know that our favorite problems have polynomial time algorithms, we can then worry about making them run in $O(n \log n)$ or $O(n^2)$ time!

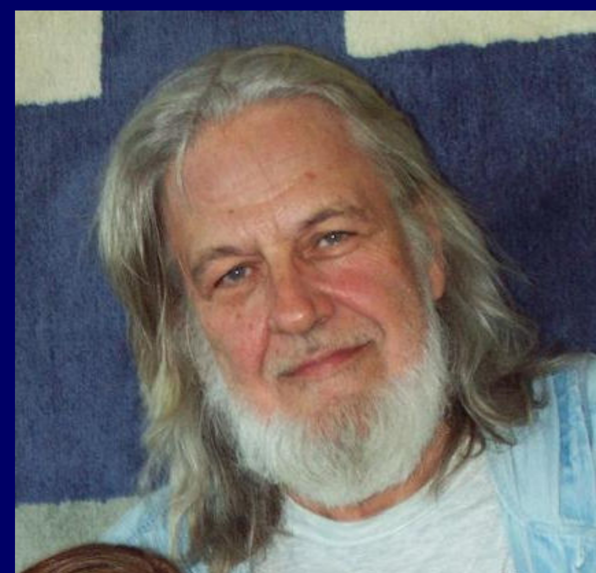
What about our favorite problems SAT / HAM / BI MATCH ?

The class P defined in the 50's.

The Intrinsic Computational Difficulty of Functions, Alan Cobham, 1964.

Paths, Trees and Flowers, Jack Edmonds, 1965.

not the correct Alan Cobham



this is indeed Jack Edmonds

Paths, Trees and Flowers, Jack Edmonds, 1965.

An explanation is due on the use of the words "efficient algorithm"…I am not prepared to set up the machinery necessary to give it formal meaning, nor is the present context appropriate for doing this…For practical purposes the difference between algebraic and exponential order is more crucial than the difference between [computable and not computable]…

It would be unfortunate for any rigid criterion to inhibit the practical development of algorithms which are either not known or known not to conform nicely to the criterion… However, if only to motivate the search for good, practical algorithms, it is important to realize that it is mathematically sensible even to question their existence.

Edmonds called them "*good algorithms*"

The Intrinsic Computational Difficulty of Functions, Alan Cobham, 1964.

For several reasons the class P seems a natural one to consider. For one thing, if we formalize the definition relative to various general classes of computing machines we seem always to end up with the same well-defined class of functions. Thus we can give a mathematical characterization of P having some confidence it characterizes correctly our informally defined class.

This class then turns out to have several natural closure properties, being closed in particular under explicit transformation, composition and limited recursion on notation (digit-by-digit recursion).

if p( ) and q( ) are polynomials, then p(q( )) is also a polynomial

The class P

We say a set $L \subseteq \Sigma^*$ is in P if there is
a program A and
a polynomial $p()$

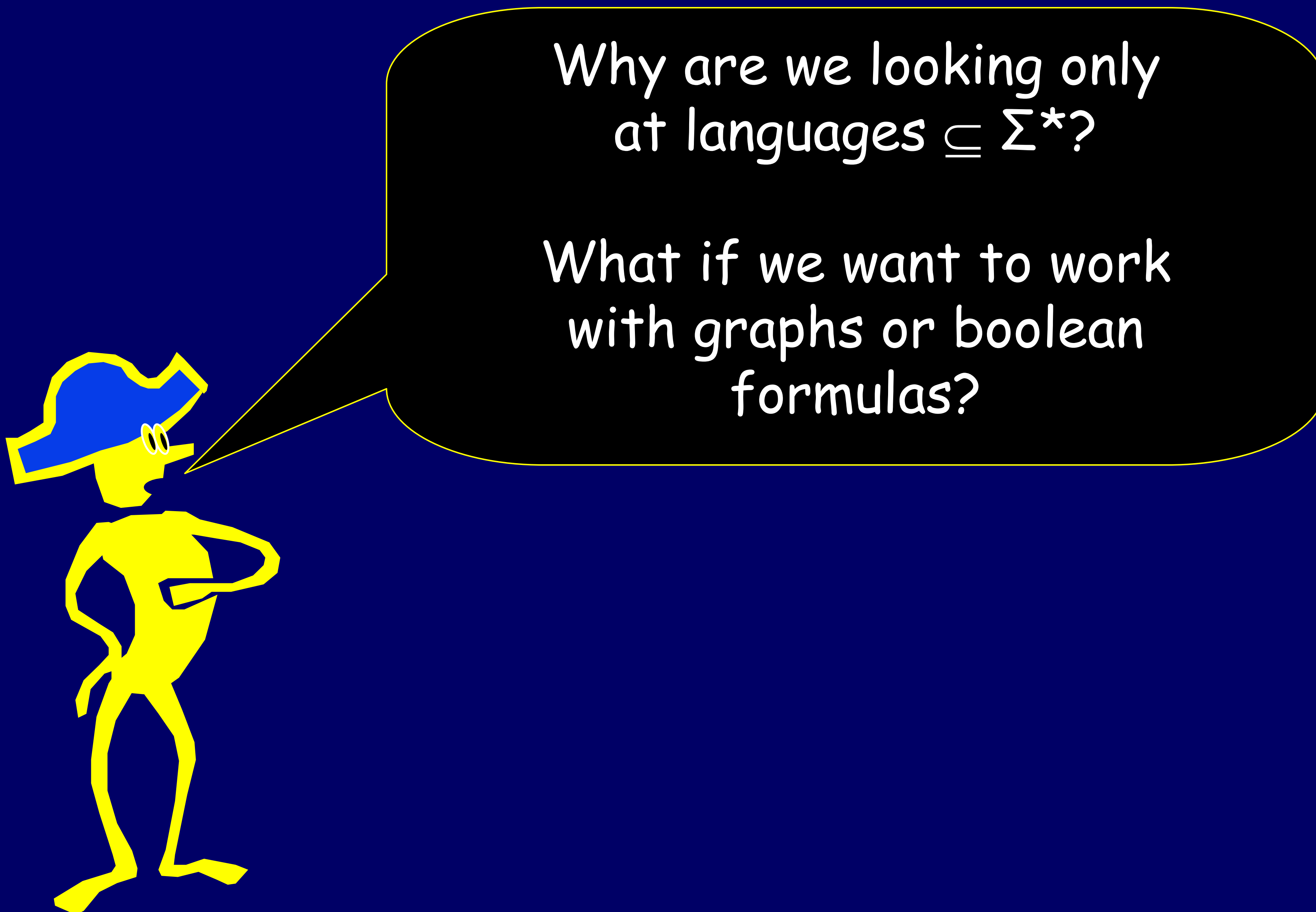
such that for any x in Σ^*,

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

The class P

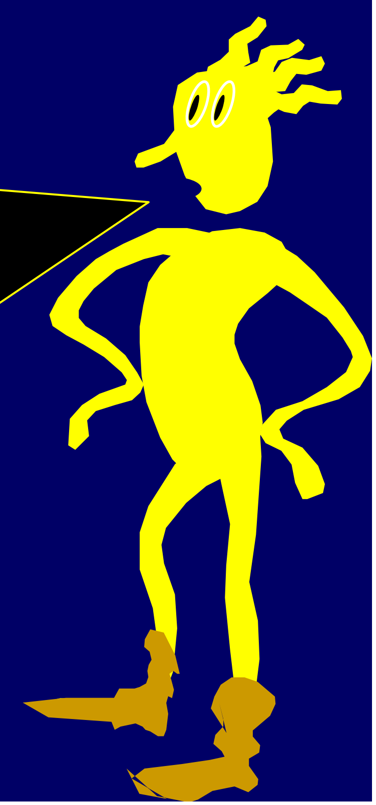
The class of all sets L that can be recognized in polynomial time.

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


Why are we looking only at languages $\subseteq \Sigma^*$?
What if we want to work with graphs or boolean formulas?

Requiring that $L \subseteq \Sigma^*$ is not really restrictive, since we can encode graphs and Boolean formulas as strings of 0's and 1's.

In fact, we do this all the time: inputs for all our programs are just sequences of 0's and 1's encoded in some suitable format.



Languages/functions in P?

<u>Example 1:</u>

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 nodes you see. If you see n nodes, $G \in$ CONN, else not.

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

Languages/functions in P?

$$\left.\begin{array}{l}\text{HAM}\\ \text{SUDOKU}\\ \text{SAT}\end{array}\right\}\ \text{not known.}$$

$$\text{BI-MATCH} \longleftarrow O(|V| \cdot \sqrt{|E|})$$

Languages/functions in P?

Co-HAM = { all graphs G that have <u>no</u> Hamilton cycle }

$G \in \text{Co-HAM} \iff G \notin \text{HAM}.$

$\text{HAM} \in P \iff \text{Co-HAM} \in P$

Onto the new class, NP

VERIFYING MEMBERSHIP

Is there a short "proof" I can give you for:

$G \in$ HAM ?

$G \in$ BI-MATCH ?

$C \in$ SAT ?

$G \in$ CO-HAM ?

NP

A set $L \in NP$

if $\exists$ an algorithm A

and a polynomial $p()$

$\forall x \in L$

$\exists y$ with $|y| \leq p(|x|)$

s.t. $A(x,y) = YES$

in $p(|x|)$ time

$\forall x' \notin L$

$\forall y'$ with $|y'| \leq p(|x'|)$

$A(x', y') = NO.$

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

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

The new class NP

We say a set $L \subseteq \Sigma^*$ is in NP if there is
a program A and a polynomial $p()$
such that for any x in Σ^*,

a short proof
that x in L

If $x \in L$, there exists a "proof" y with $|y| \le p(|x|)$
$A(x, y)$ runs for $\le p(|x|)$ time
and answers "x is in L" correctly.

that can
be quickly
"verified"

If $x \notin L$, for all "proofs" y
$A(x, y)$ answers "x not in L" correctly.

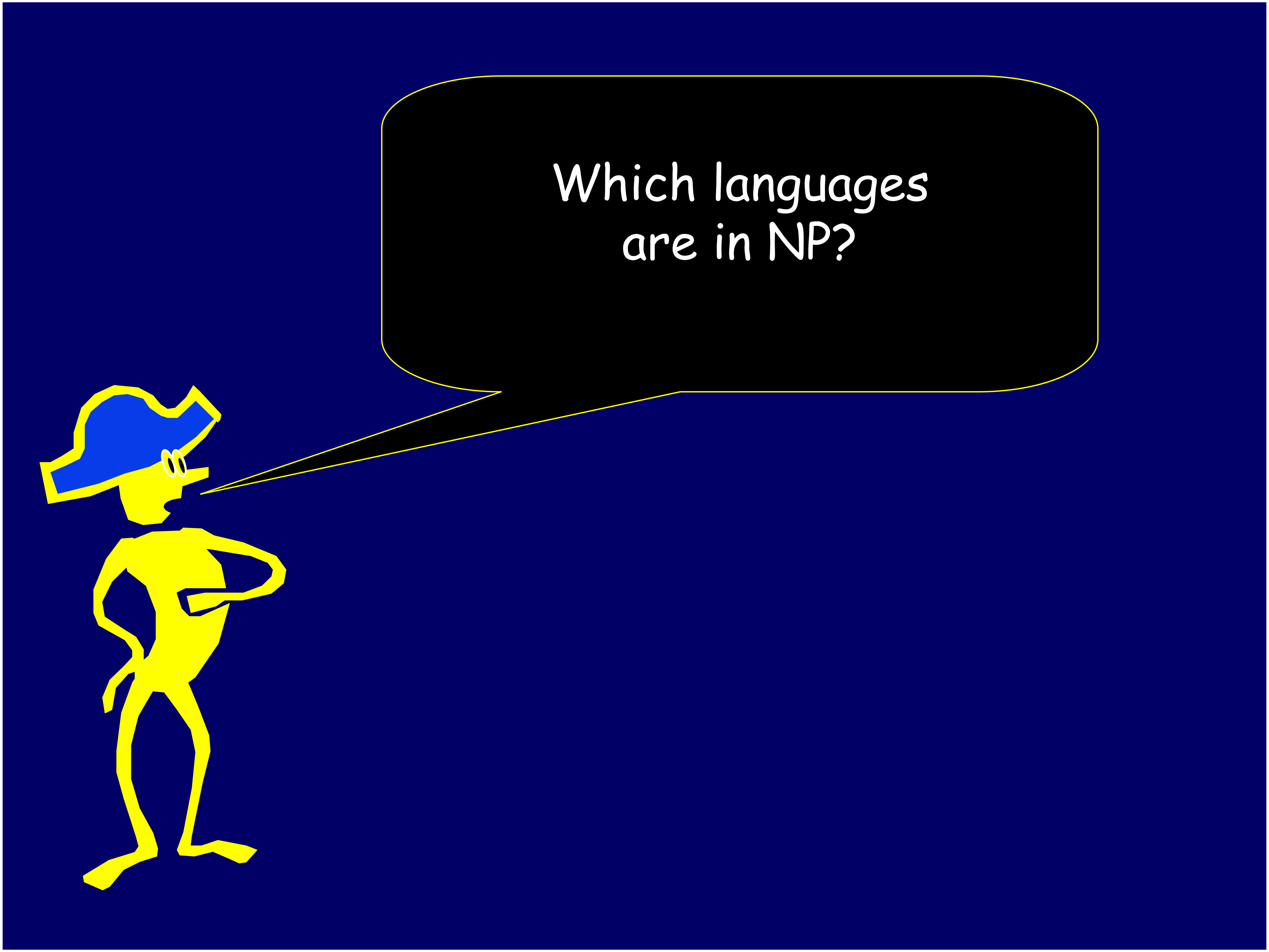
Verifier rejects all "fake" proofs

The class NP

Non-deterministic polynomial time.

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

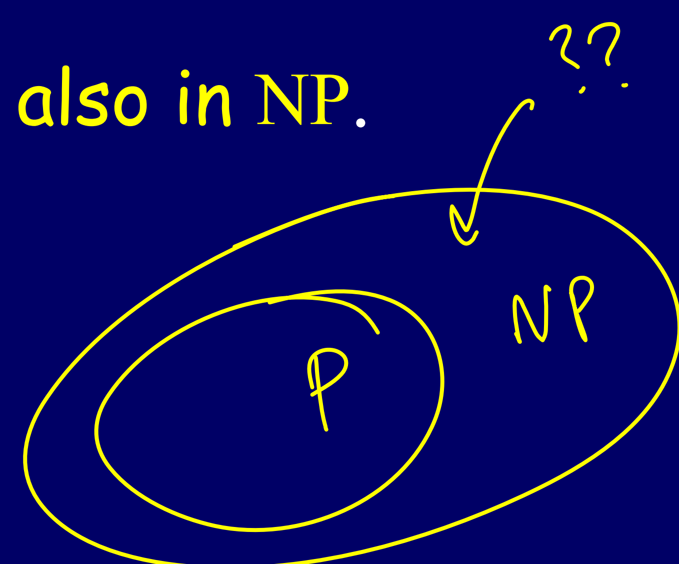
Recall: A doesn't have to find these proofs y; it just needs to be able to verify that y is a "correct" proof.


Which languages
are in NP?

P ⊆ 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?

HAM? ✓

SAT? ✓

BI-MATCH? ✓

SUDOKU? ✓

non-hamiltonian ~~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 $NP \subseteq P$?

The P versus NP problem

If membership in L can be <u>verified</u> in poly-time $\Big\}$ $L \in NP$

$\Downarrow$

can membership in L be decided in polytime? $\Big\}$ $L \in P$

??

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

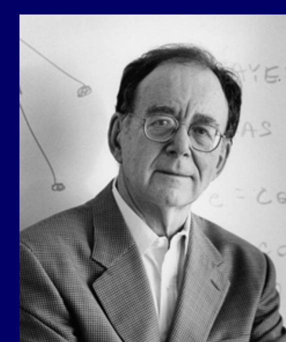
Allocating variables to registers

Finding good packet routings in networks

Decryption

…

Hence proving P = NP would break cryptosystems



OK, OK, I CARE.

WHERE DO I BEGIN?

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

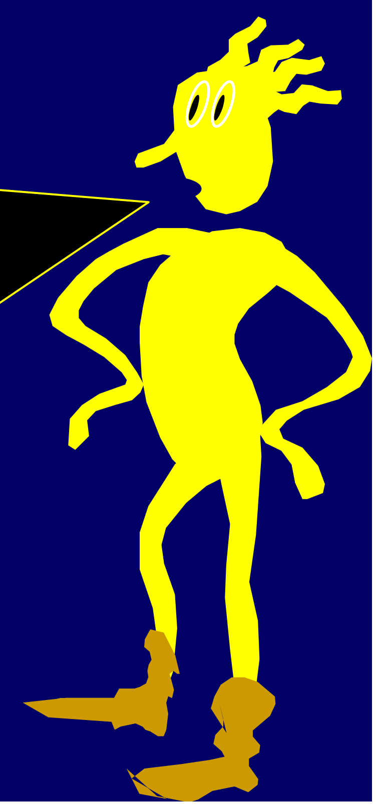
I would have to show that
<u>every</u> set in NP has a
polynomial time algorithm…

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

Relax, Bonzo!

We can describe
<u>one</u> 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.

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	

SUDOKU

$$= \bigcup_{n \in \mathbb{Z}} \{\text{all solvable } n \times n \times n \text{ Sudoku instances}\}$$

SUDOKU has polynomial-time algorithm

$\Longleftrightarrow$

all of NP does

THE "HARDEST" SETS in NP

SUDOKU

3COLOR

CLIQUE

SAT

INDEPENDENT-SET

HAM

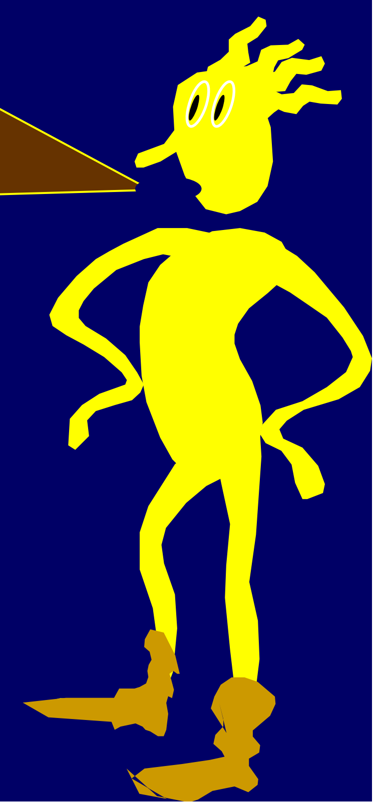
HOW DO YOU PROVE
THESE ARE
THE HARDEST?

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.

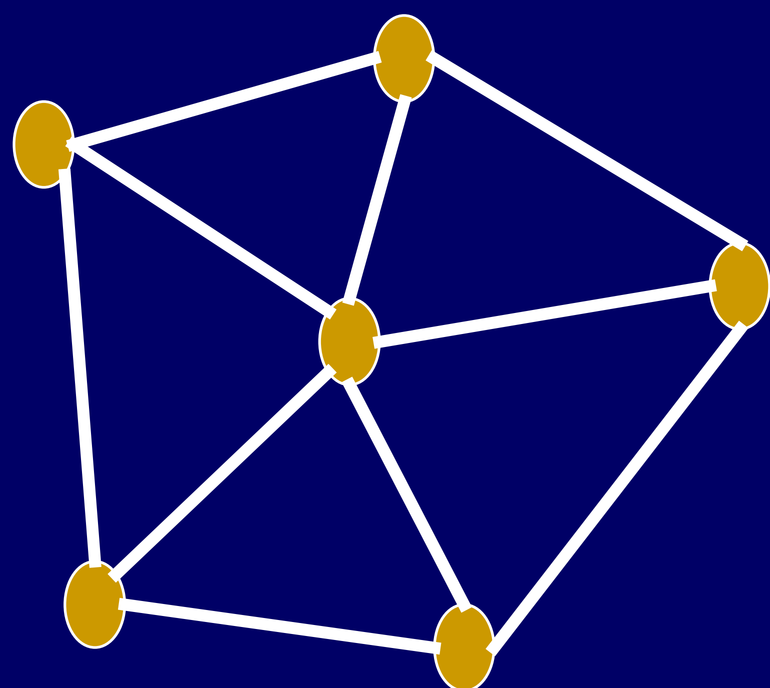
We say SAT is **NP-complete**.



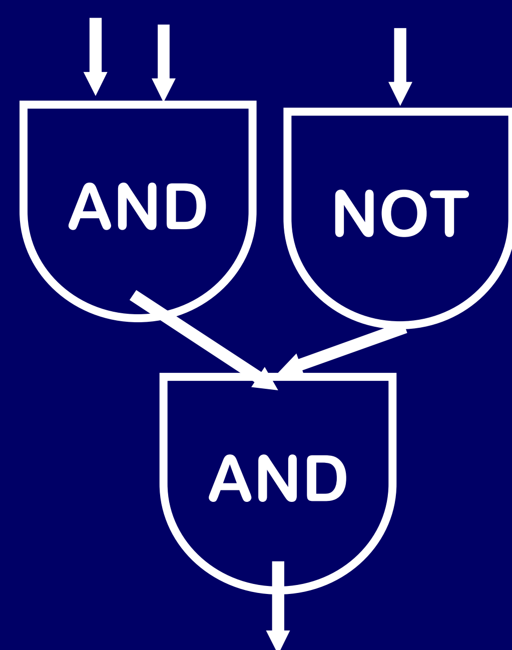
Proof not difficult, but we defer the proof to 15-451 (Algos) or 15-453 (FLAC)

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.

Any language in NP

can be reduced
(in polytime to)
an instance of

[Cook | Levin]

SAT

hence SAT is NP-complete

Any language in NP

can be reduced
(in polytime to)
an instance of

SAT

can be reduced
(in polytime to)
an instance of

3COLOR

Hence:

Any language in NP

↓ can be reduced (in polytime to) an instance of

SAT

↓ can be reduced (in polytime to) an instance of

3COLOR

hence 3COLOR is NP-complete

To show a language L is NP-complete

To show L is NP-complete, we must

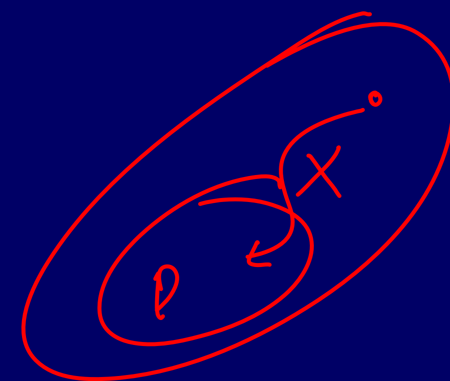
a) Show L is in NP
b) Reduce all problems in NP to L

or

b) Reduce some NP-complete language (e.g. SAT) to L
(by exhibiting a function F such that
C is satisfiable $\Leftrightarrow$ F(C) is in L)

NP-complete problems

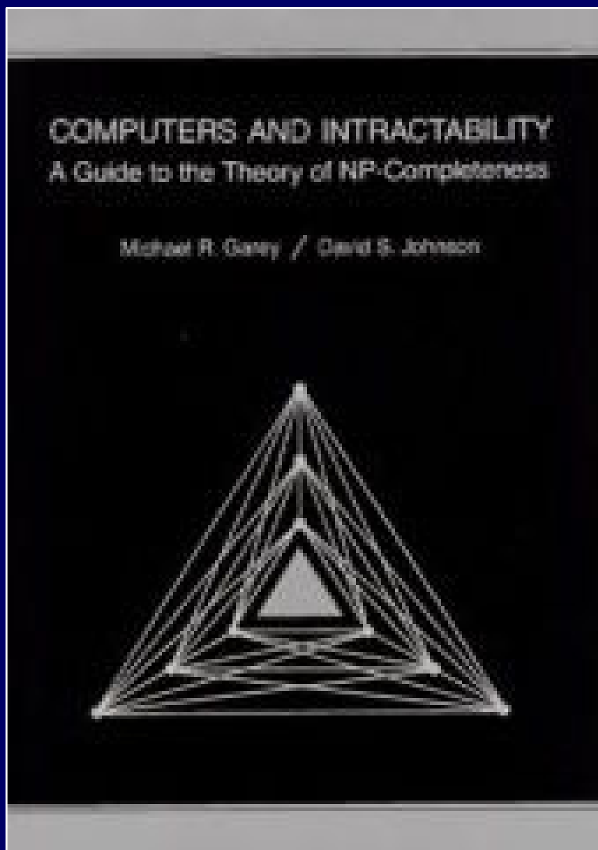
3COLOR, SAT, bin-packing, classroom scheduling,
many other problems NP-complete.
proofs follow above pattern.



NP-complete problems: "Hardest" problems in NP
(if any of these problems in P, then all NP in P.)

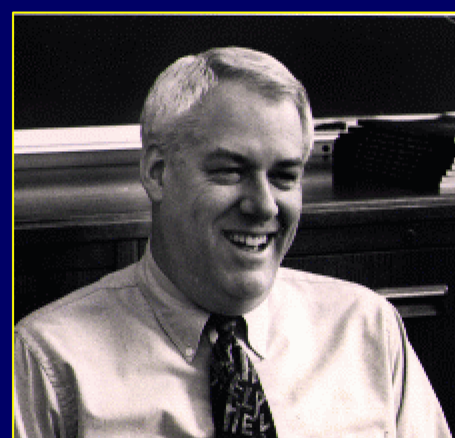
NP-complete problems come up all the time
Many problems of practical importance are NP-complete
If P $\neq$ NP, we will not have poly-time algorithms for them.
Right now, <u>we don't know</u>.

Reference



Computers and Intractablity:
A guide to the Theory of NP-completeness

by Mike Garey and David Johnson



References

The NP-completeness Column:
1981-2005
by David Johnson
http://www.research.att.com/~dsj/columns/

The P versus NP problem
Official Description, Clay Institute
by Steve Cook
http://www.claymath.org/millennium/P_vs_NP/