BBM 202 - ALGORITHMS

HACETTEPE UNIVERSITY

DEPT. OF COMPUTER ENGINEERING

REDUCTIONS
INTRACTABILITY

Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.

Overview

Main topics.

- Reduction: design algorithms, establish lower bounds, classify problems.
- Intractability: problems beyond our reach.

Shifting gears.

- From individual problems to problem-solving models.
- From linear/quadratic to polynomial/exponential scale.
- From details of implementation to conceptual framework.

Goals.

- Place algorithms we've studied in a larger context.
- Introduce you to important and essential ideas.
- Inspire you to learn more about algorithms!

Advanced Topics

- Reductions
- Designing algorithms
- Establishing lower bounds
- Classifying problems

- Intractability

Bird's-eye view

Desiderata. Classify problems according to computational requirements.

complexity	order of growth	examples
linear	N	min, max, median, Burrows-Wheeler transform, …
linearithmic	$N \log N$	sorting, convex hull, closest pair, farthest pair, …
quadratic	N^2	?
⋮	⋮	⋮
exponential	c^N	?

Frustrating news. Huge number of problems have defied classification.

Bird's-eye view

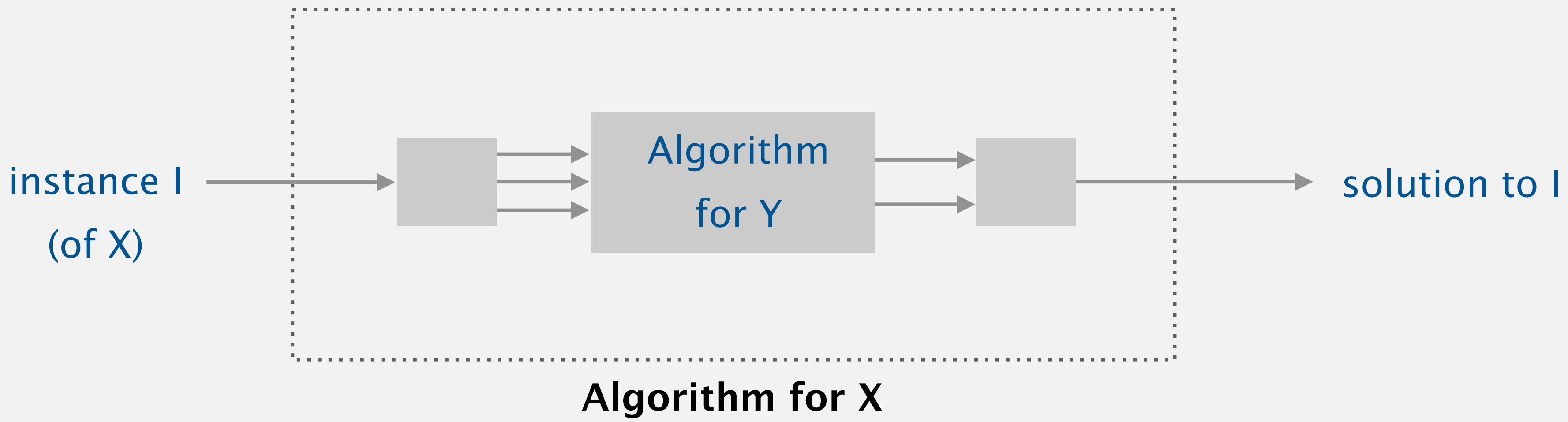
Desiderata. Classify problems according to computational requirements.

Desiderata'.
Suppose we could (could not) solve problem X efficiently.
What else could (could not) we solve efficiently?

" Give me a lever long enough and a fulcrum on which to place it, and I shall move the world. " — *Archimedes*

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



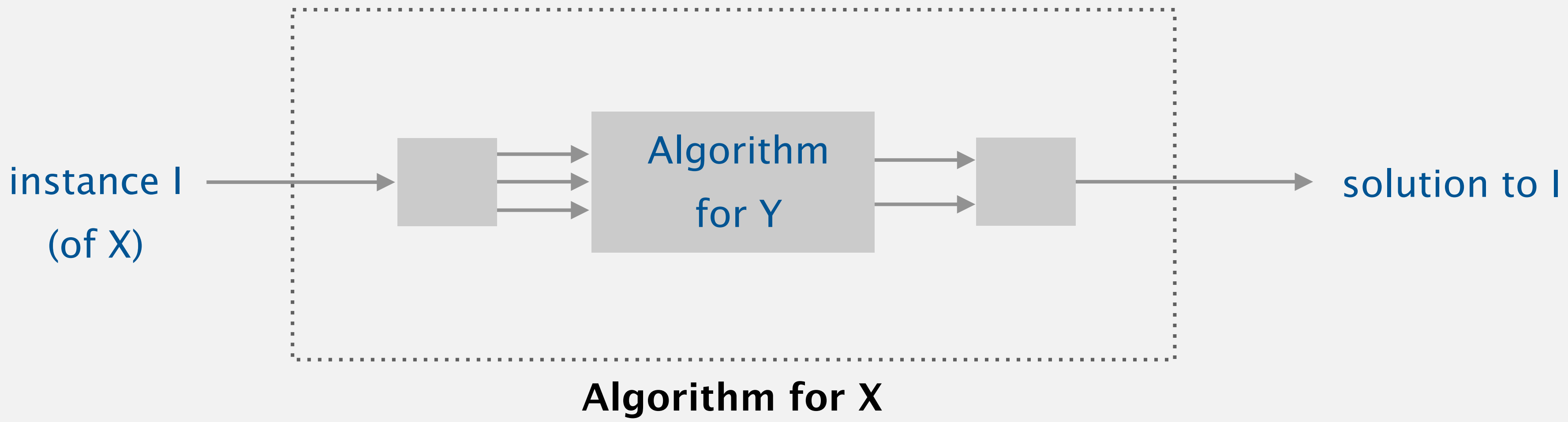
Cost of solving X = total cost of solving Y + cost of reduction.

perhaps many calls to Y
on problems of different sizes

preprocessing and postprocessing

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Ex 1. [finding the median reduces to sorting]

To find the median of N items:

- Sort N items.
- Return item in the middle.

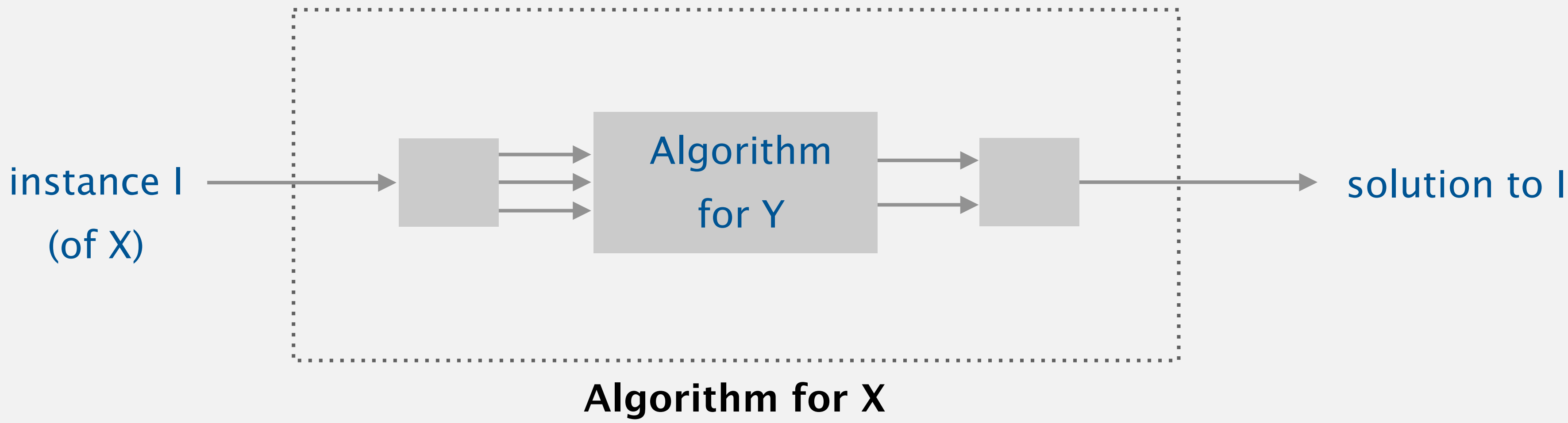
cost of sorting

cost of reduction

Cost of solving element distinctness. $N \log N + 1$.

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Ex 2. [element distinctness reduces to sorting]

To solve element distinctness on N items:

- Sort N items.
- Check adjacent pairs for equality.

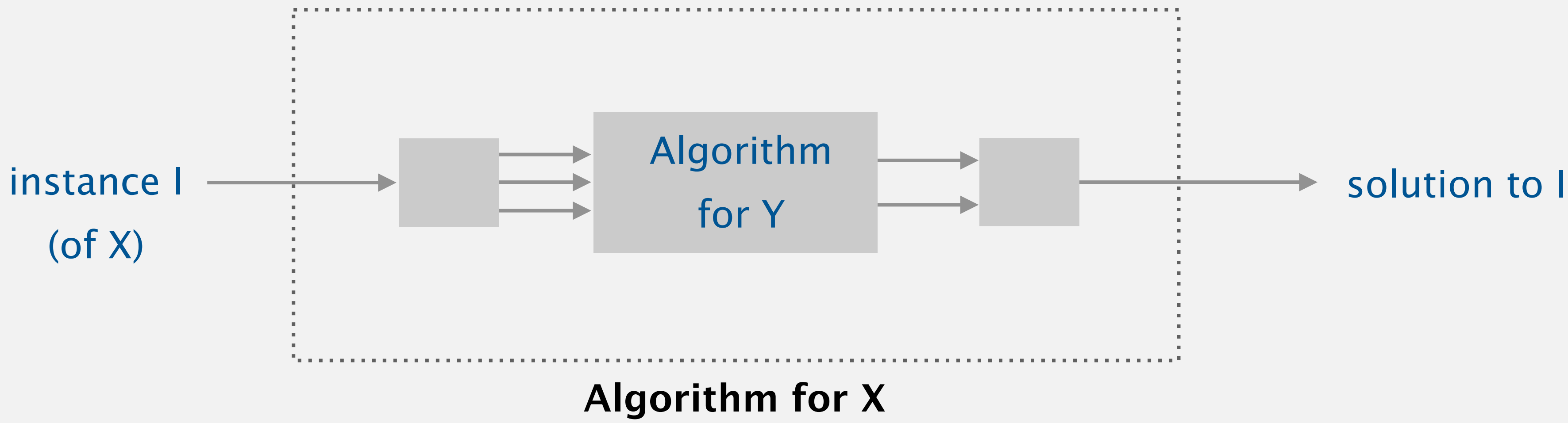
cost of sorting

cost of reduction

Cost of solving element distinctness. $N \log N + N$.

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Ex 3. [3-collinear reduces to sorting]

To solve 3-collinear instance on N points in the plane:

- For each point, sort other points by polar angle or slope.
  - check adjacent triples for collinearity

cost of sorting

cost of reduction

Cost of solving 3-collinear. $\underbrace{N^2 \log N}_{\text{cost of sorting}} + \underbrace{N^2}_{\text{cost of reduction}}$.

Reductions

- Designing algorithms
- Establishing lower bounds
- Classifying problems

Reduction: design algorithms

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.

Design algorithm. Given algorithm for Y, can also solve X.

Ex.

- Element distinctness reduces to sorting.
- 3-collinear reduces to sorting.
- CPM reduces to topological sort. [shortest paths lecture]
- h-v line intersection reduces to 1d range searching. [geometric BST lecture]
- Baseball elimination reduces to maxflow.
- Burrows-Wheeler transform reduces to suffix sort.
- …

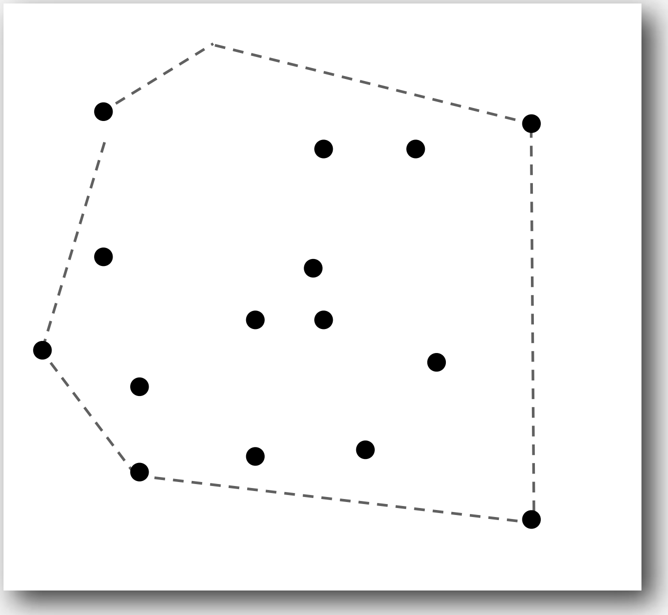
Mentality. Since I know how to solve Y, can I use that algorithm to solve X?

programmer's version: I have code for Y. Can I use it for X?

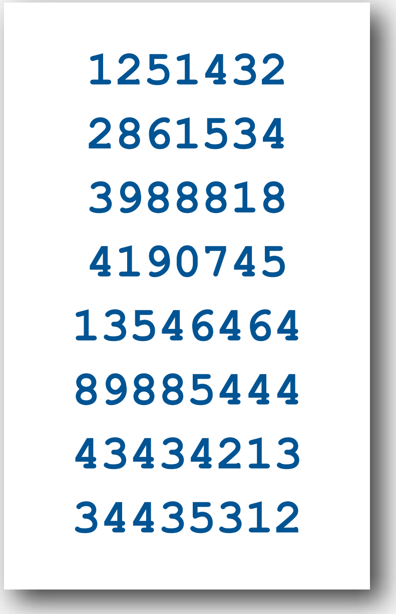
Convex hull reduces to sorting

Sorting. Given N distinct integers, rearrange them in ascending order.

Convex hull. Given N points in the plane, identify the extreme points of the convex hull (in counterclockwise order).



convex hull



sorting

Proposition. Convex hull reduces to sorting.

Pf. Graham scan algorithm.

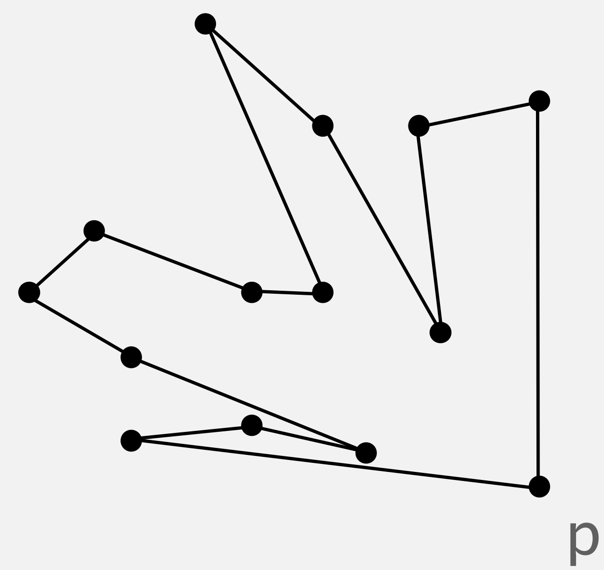
cost of sorting   cost of reduction

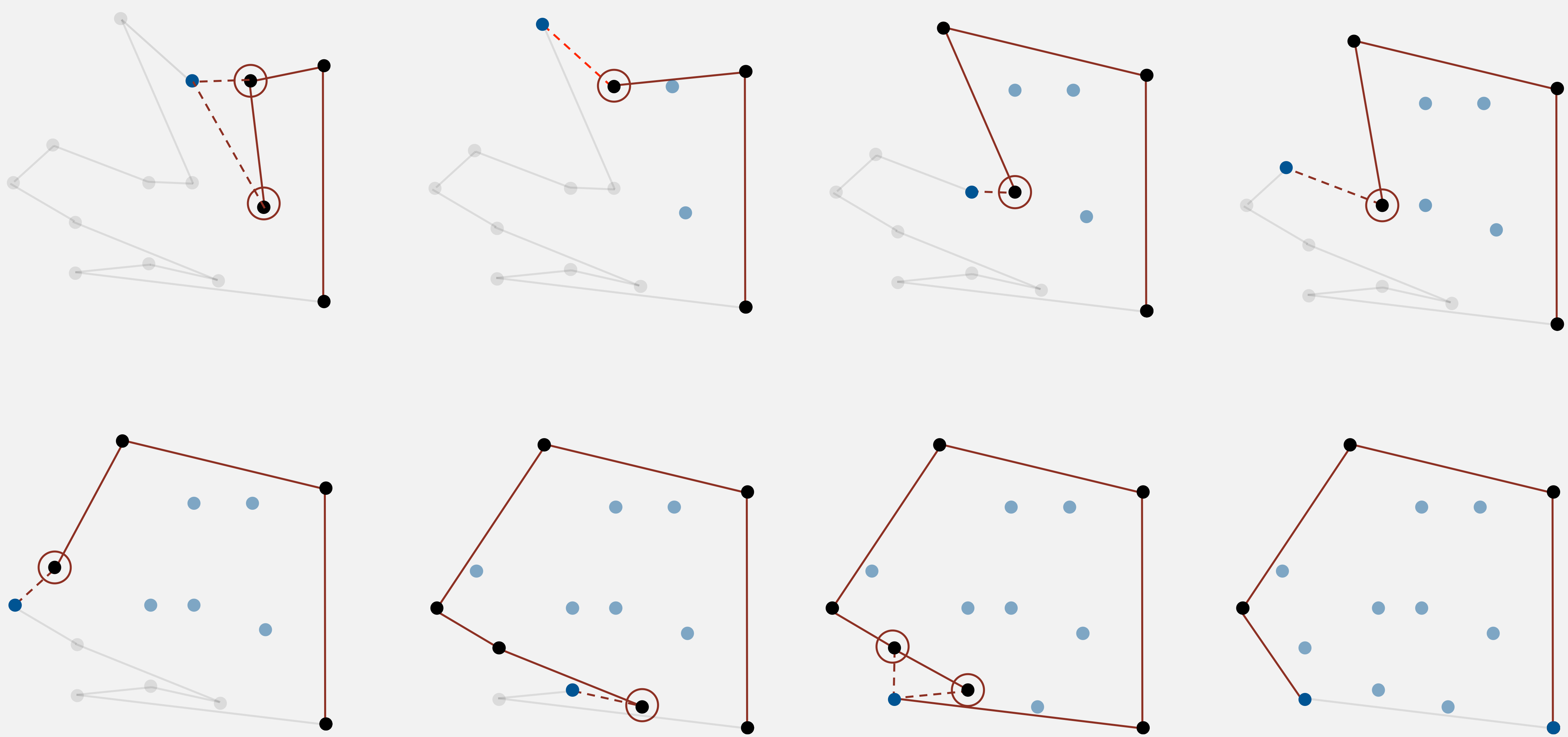
Cost of convex hull. $N \log \mathrm{N} \; + \; N$.

Graham scan algorithm

Graham scan.

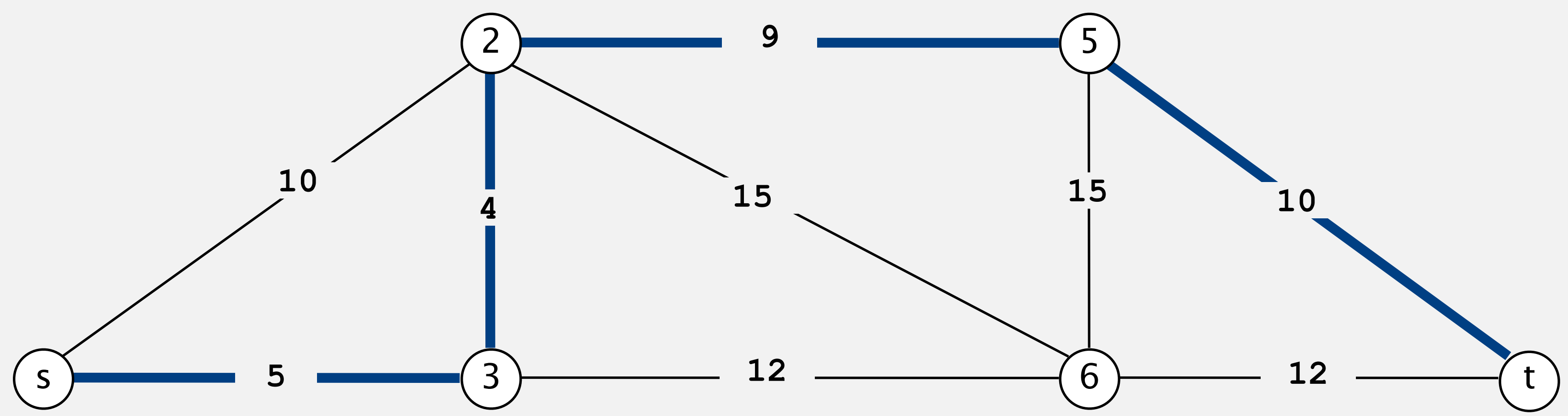
- Choose point p with smallest (or largest) y-coordinate.
- Sort points by polar angle with p to get simple polygon.
- Consider points in order, and discard those that would create a clockwise turn.





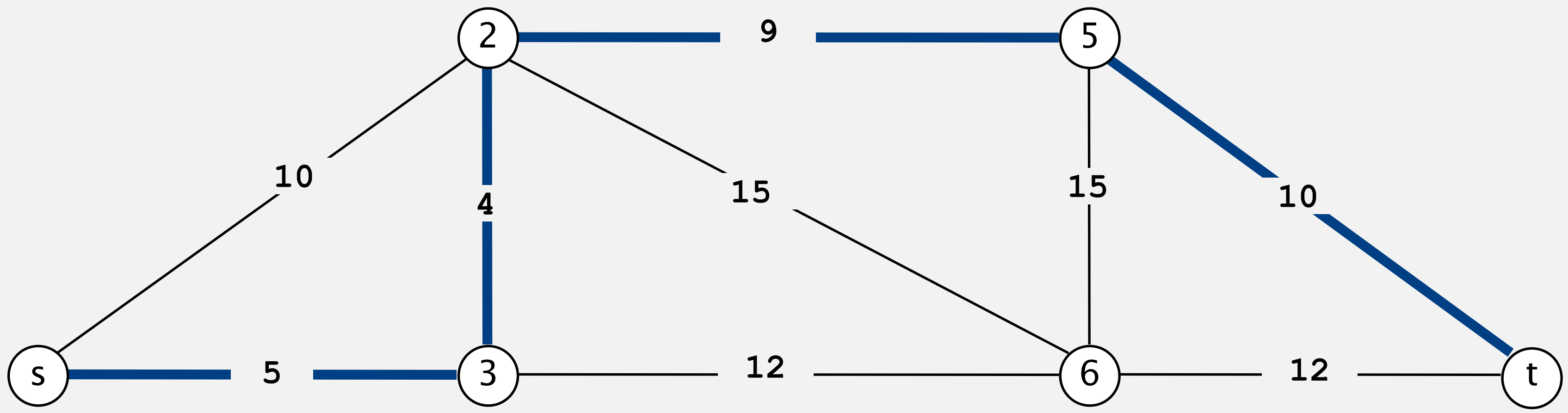
Shortest paths on edge-weighted graphs and digraphs

Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.

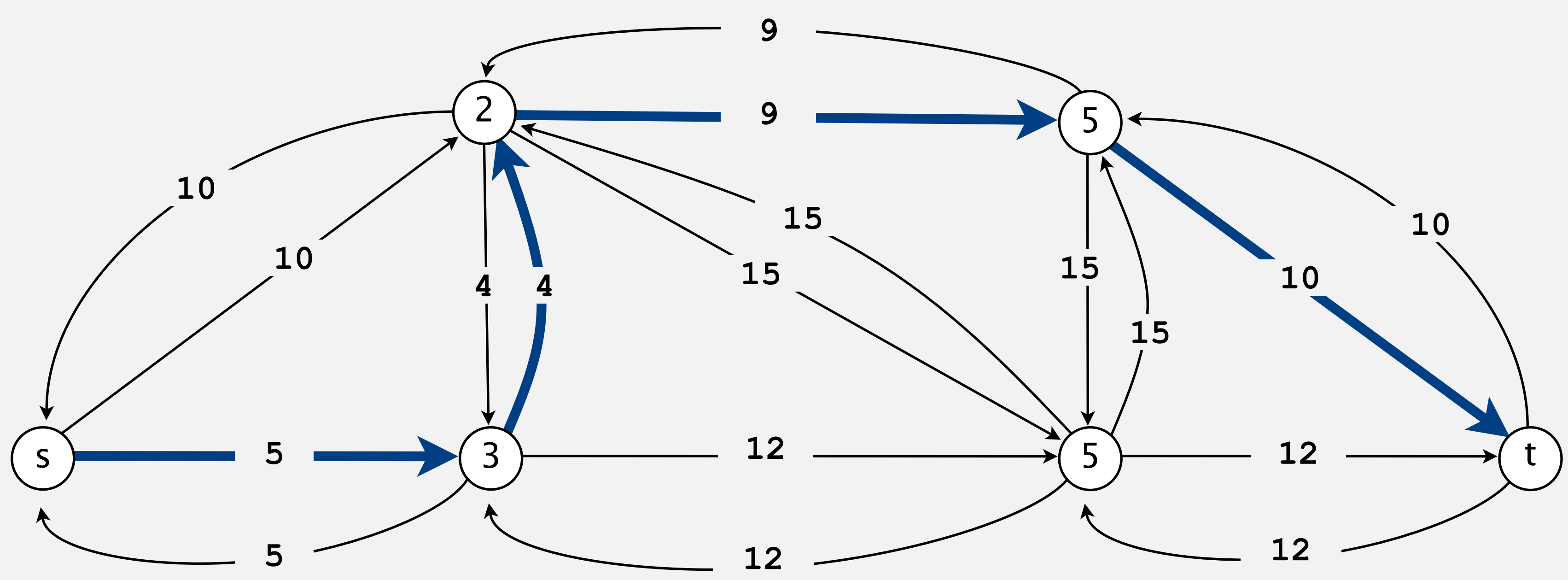


Shortest paths on edge-weighted graphs and digraphs

Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.

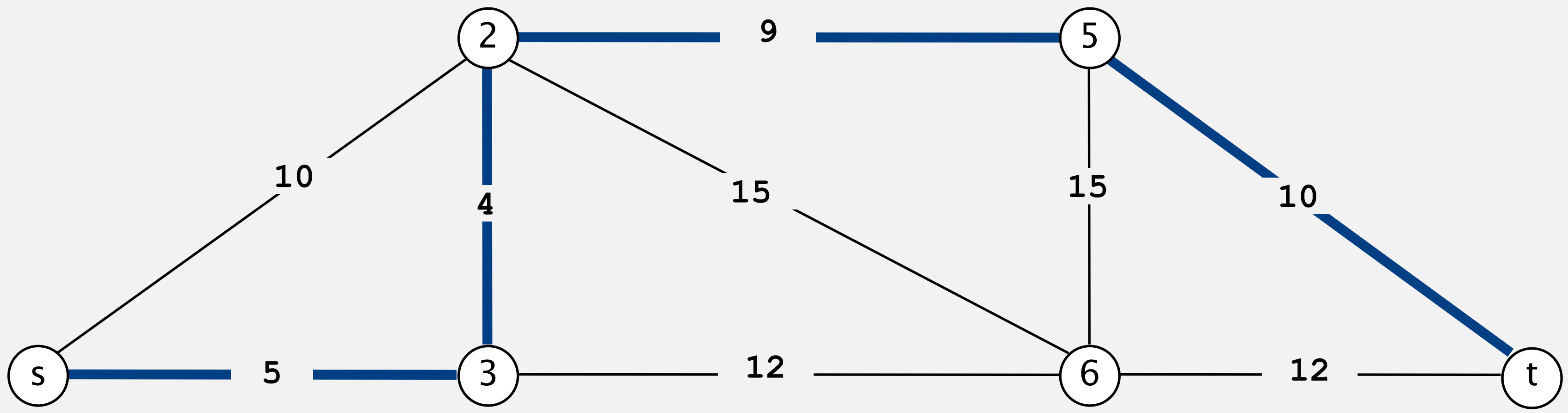


Pf. Replace each undirected edge by two directed edges.



Shortest paths on edge-weighted graphs and digraphs

Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.



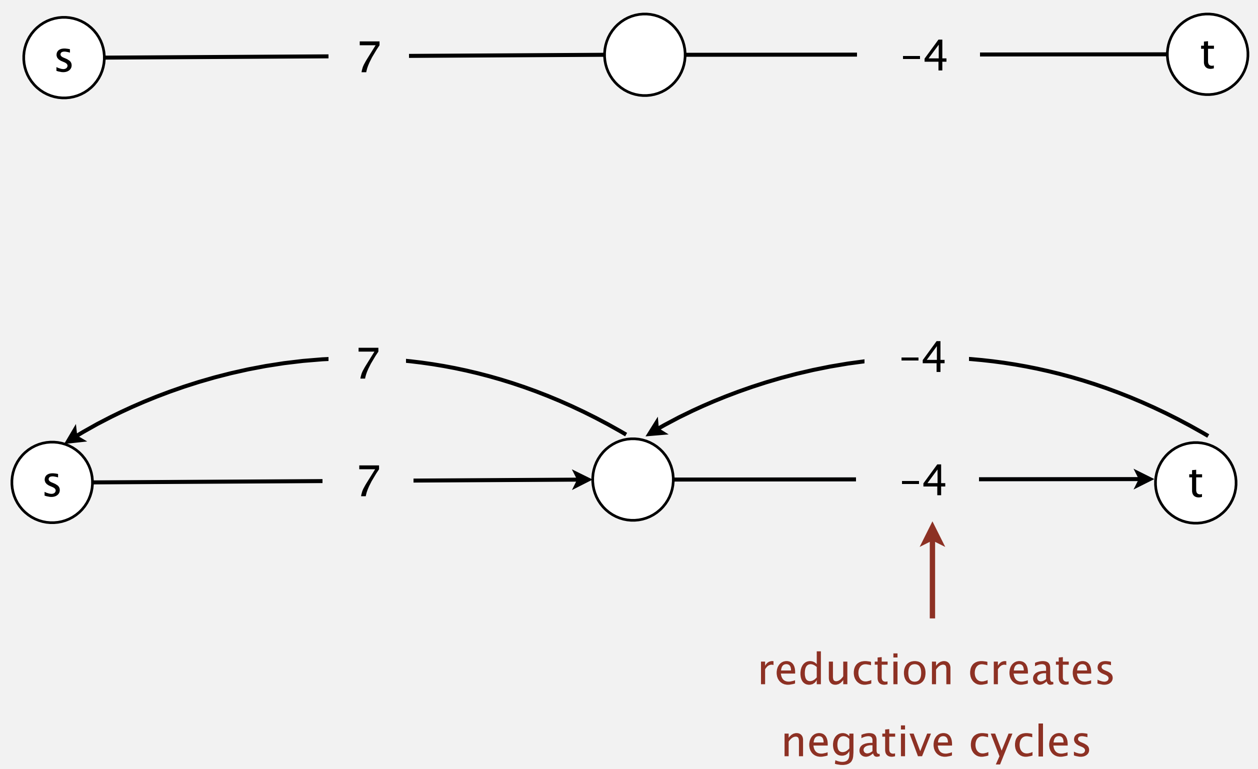
cost of shortest paths in digraph

cost of reduction

Cost of undirected shortest paths. $E \log V + E$.

Shortest paths with negative weights

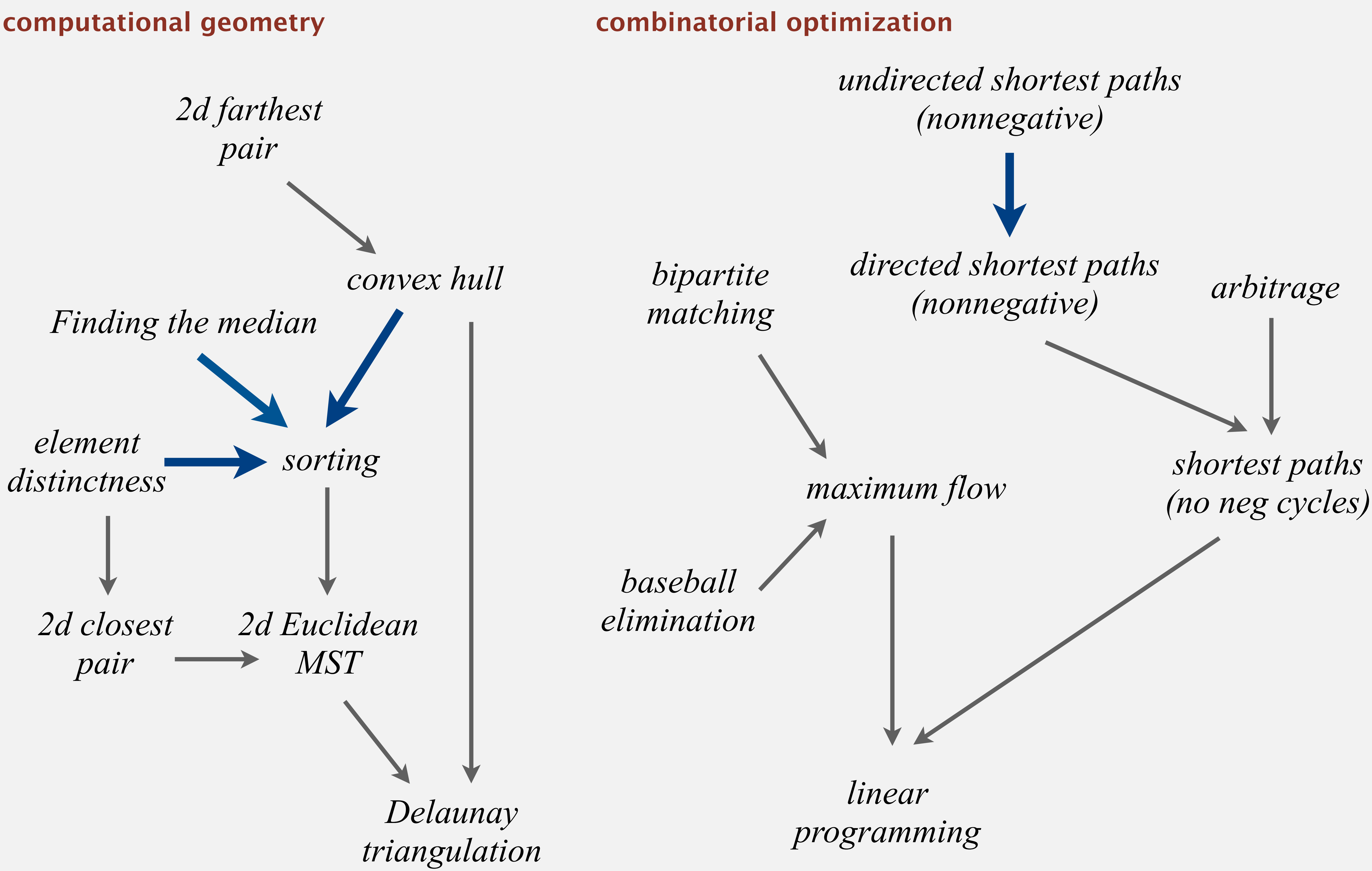
Caveat. Reduction is invalid for edge-weighted graphs with negative weights (even if no negative cycles).



Remark. Can still solve shortest-paths problem in undirected graphs (if no negative cycles), but need more sophisticated techniques.

reduces to weighted
non-bipartite matching (!)

Some reductions involving familiar problems



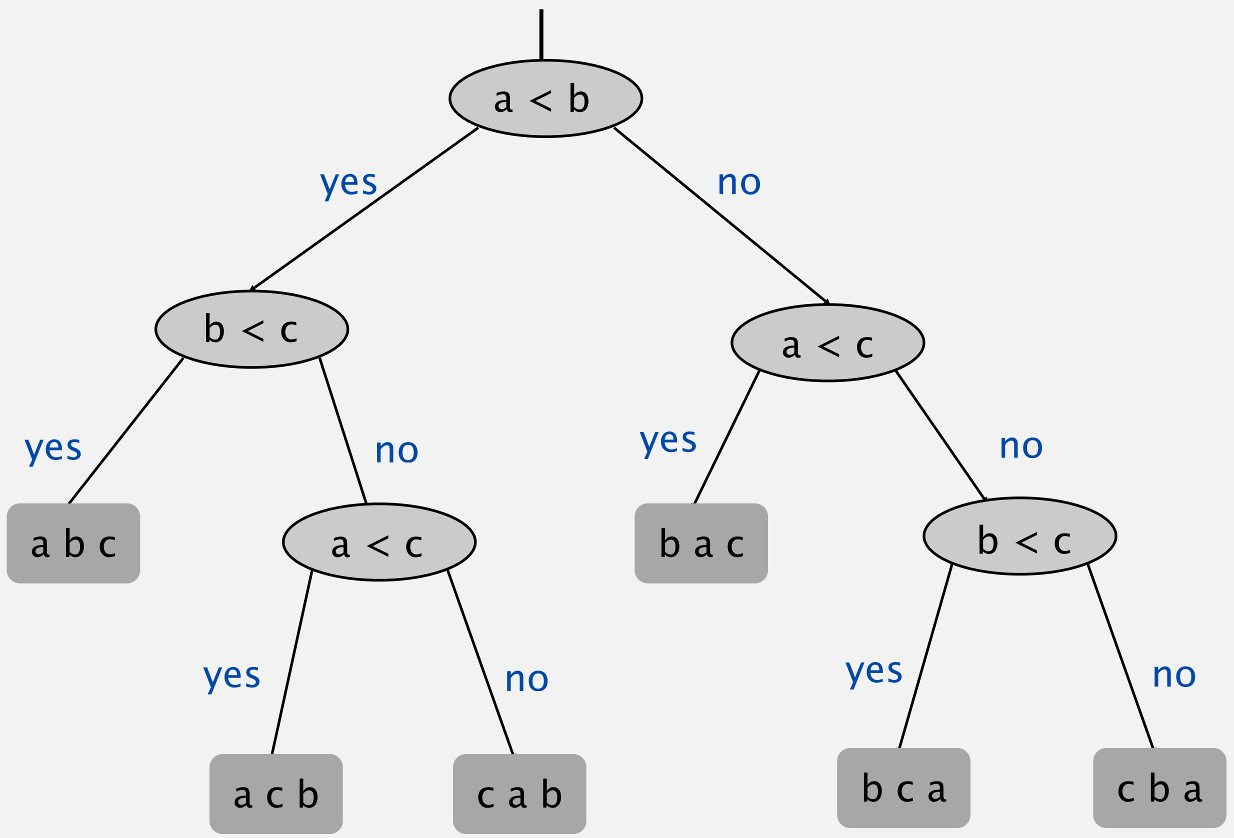
Reductions

- Designing algorithms
- Establishing lower bounds
- Classifying problems

Bird's-eye view

Goal. Prove that a problem requires a certain number of steps.

Ex. In decision tree model, any compare-based sorting algorithm requires $\Omega(N \log N)$ compares in the worst case.



argument must apply to all conceivable algorithms

Bad news. Very difficult to establish lower bounds from scratch.

Good news. Spread $\Omega(N \log N)$ lower bound to Y by reducing sorting to Y.

assuming cost of reduction is not too high

Linear-time reductions

Def. Problem X linear-time reduces to problem Y if X can be solved with:

- Linear number of standard computational steps.
- Constant number of calls to Y.

Ex. Almost all of the reductions we've seen so far.

Establish lower bound:

- If X takes $\Omega(N \log N)$ steps, then so does Y.
- If X takes $\Omega(N^2)$ steps, then so does Y.

Mentality.

- If I could easily solve Y, then I could easily solve X.
- I can't easily solve X.
- Therefore, I can't easily solve Y.

Element distinctness linear-time reduces to closest pair

Closest pair. Given N points in the plane, find the closest pair.

Element distinctness. Given N elements, are any two equal?

Proposition. Element distinctness linear-time reduces to closest pair.

Pf.

- Element distinctness instance: $x_1, x_2, \ldots, x_N$.
- Closest pair instance: $(x_1, x_1), (x_2, x_2), \ldots, (x_N, x_N)$.
- Two elements are distinct if and only if closest pair = 0.

allows quadratic tests of the form:

$x_i < x_j$ or $(x_i - x_k)^2 - (x_j - x_k)^2 < 0$

Element distinctness lower bound. In quadratic decision tree model, any algorithm that solves element distinctness takes $\Omega(N \log N)$ steps.

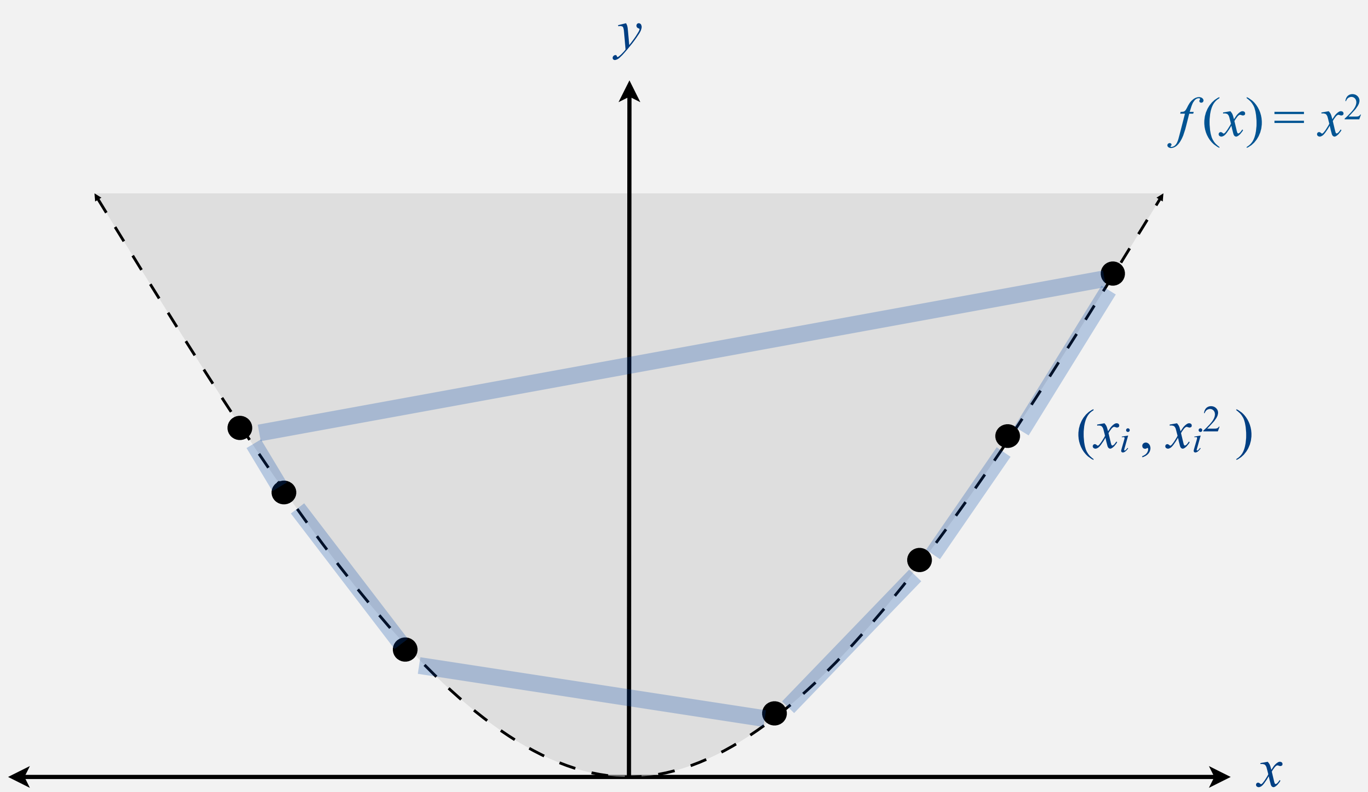
Implication. In quadratic decision tree model, any algorithm for closest pair takes $\Omega(N \log N)$ steps.

Sorting linear-time reduces to convex hull

Proposition. Sorting linear-time reduces to convex hull.

- Sorting instance: $x_1, x_2, \ldots, x_N$.
- Convex hull instance: $(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_N, x_N^2)$.

lower-bound mentality:
if I can solve convex hull
efficiently, I can sort efficiently

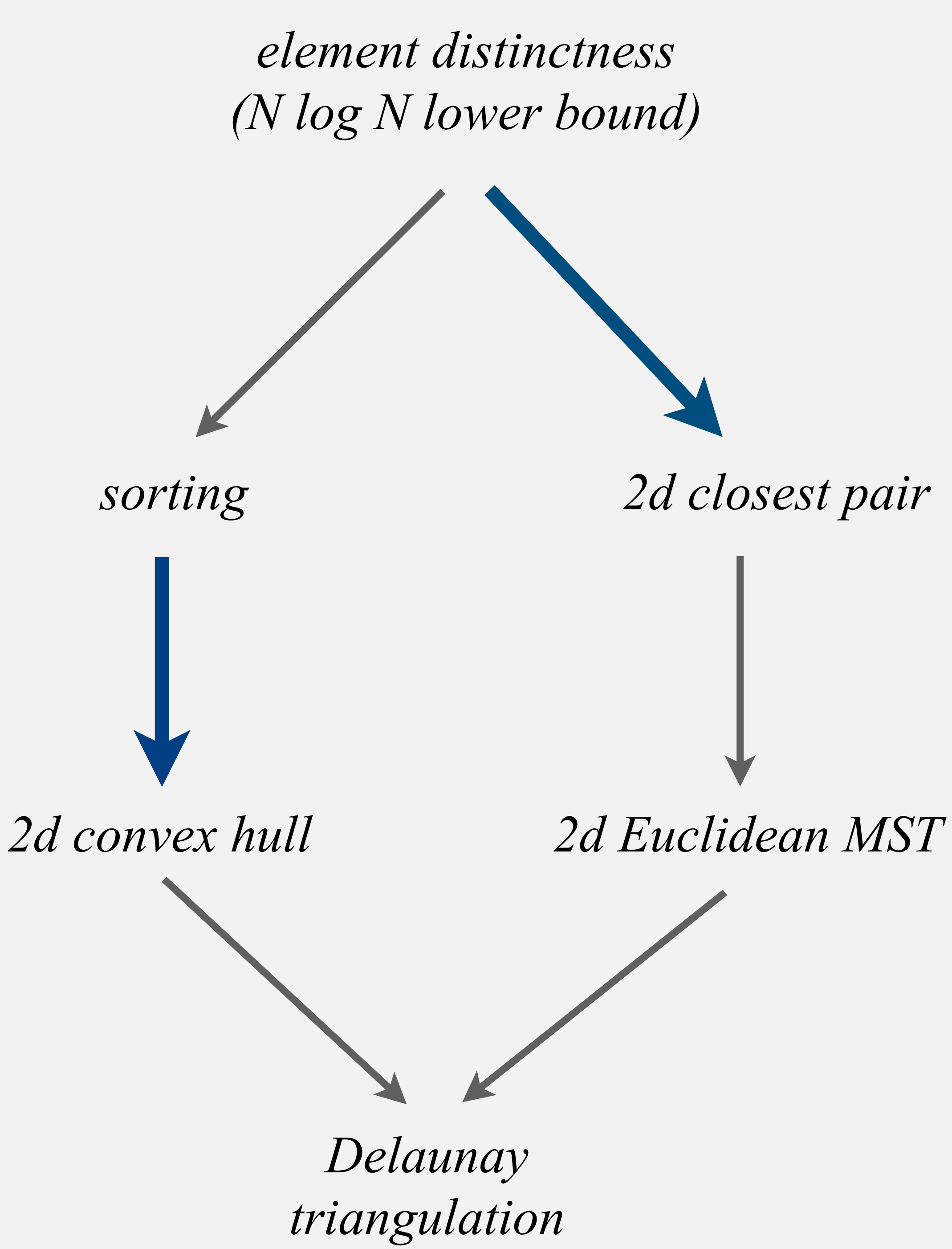


Pf.

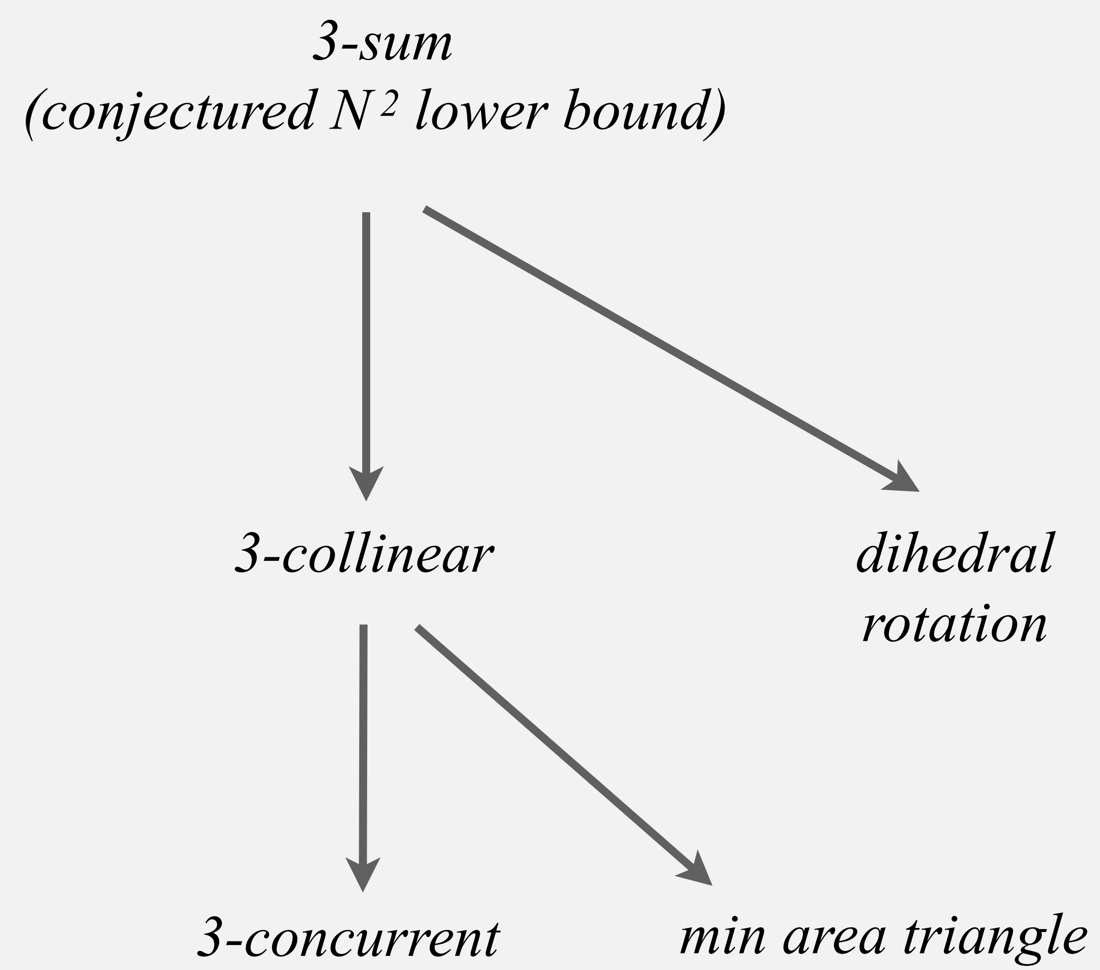
- Region $\{ x : x^2 \geq x \}$ is convex $\Rightarrow$ all points are on hull.
- Starting at point with most negative x, counterclockwise order of hull points yields integers in ascending order.

More linear-time reductions and lower bounds

sorting



3-sum



Establishing lower bounds: summary

Establishing lower bounds through reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself no linear-time convex hull algorithm exists?

A1. [hard way] Long futile search for a linear-time algorithm.

A2. [easy way] Linear-time reduction from sorting.

Reductions

- Designing algorithms
- Establishing lower bounds
- Classifying problems

Classifying problems: summary

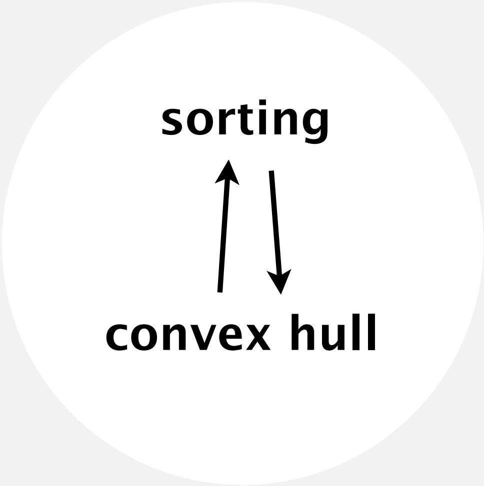
Desiderata. Problem with algorithm that matches lower bound.

Ex. Sorting, convex hull, and closest pair have complexity $N \log N$.

Desiderata'. Prove that two problems X and Y have the same complexity.

- First, show that problem X linear-time reduces to Y.
- Second, show that Y linear-time reduces to X.
- Conclude that X and Y have the same complexity.

even if we don't know what it is!



Caveat

SORT. Given N distinct integers, rearrange them in ascending order.

CONVEX HULL. Given N points in the plane, identify the extreme points of the convex hull (in counterclockwise order).

Proposition. *SORT* linear-time reduces to *CONVEX HULL*.

Proposition. *CONVEX HULL* linear-time reduces to *SORT*.

Conclusion. *SORT* and *CONVEX HULL* have the same complexity.

A possible real-world scenario.

- System designer specs the APIs for project.
- Alice implements `sort()` using `convexHull()`. well, maybe not so realistic
- Bob implements `convexHull()` using `sort()`.
- Infinite reduction loop!
- Who's fault?

Integer arithmetic reductions

Integer multiplication. Given two N-bit integers, compute their product.

Brute force. N^2 bit operations.

								1	1	0	1	0	1	0	1
							×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1
							0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

Integer arithmetic reductions

Integer multiplication. Given two N-bit integers, compute their product.

Brute force. N^2 bit operations.

problem	arithmetic	order of growth
integer multiplication	$a \times b$	M(N)
integer division	a / b, $a \bmod b$	M(N)
integer square	a^2	M(N)
integer square root	$\lfloor \sqrt{a} \rfloor$	M(N)

integer arithmetic problems with the same complexity as integer multiplication

Q. Is brute-force algorithm optimal?

History of complexity of integer multiplication

year	algorithm	order of growth
?	brute force	N^2
1962	Karatsuba-Ofman	$N^{1.585}$
1963	Toom-3, Toom-4	$N^{1.465}$, $N^{1.404}$
1966	Toom-Cook	$N^{1+\varepsilon}$
1971	Schönhage–Strassen	$N \log N \log \log N$
2007	Fürer	$N \log N \, 2^{\log^* N}$
?	?	N

number of bit operations to multiply two N-bit integers

used in Maple, Mathematica, gcc, cryptography, ...

Remark. GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.

GMP
«Arithmetic without limitations»

Linear algebra reductions

Matrix multiplication. Given two N-by-N matrices, compute their product.

Brute force. N^3 flops.

$$
\begin{array}{c}
\\
\text{row i}\\
\\
\\
\end{array}
\begin{bmatrix}
0,1 & 0,2 & 0,8 & 0,1 \\
0,5 & 0,3 & 0,9 & 0,6 \\
0,1 & 0 & 0,7 & 0,4 \\
0 & 0,3 & 0,3 & 0,1
\end{bmatrix}
\times
\begin{array}{c}
\begin{array}{cccc} & & \text{column j} & \end{array}\\
\begin{bmatrix}
0,4 & 0,3 & 0,1 & 0,1 \\
0,2 & 0,2 & 0 & 0,6 \\
0 & 0 & 0,4 & 0,5 \\
0,8 & 0,4 & 0,1 & 0,9
\end{bmatrix}
\end{array}
=
\begin{array}{c}
\\
\text{i}\\
\\
\\
\end{array}
\begin{array}{c}
\begin{array}{cccc} & & \text{j} & \end{array}\\
\begin{bmatrix}
0,16 & 0,11 & 0,34 & 0,62 \\
0,74 & 0,45 & 0,47 & 1,22 \\
0,36 & 0,19 & 0,33 & 0,72 \\
0,14 & 0,1 & 0,13 & 0,42
\end{bmatrix}
\end{array}
$$

$$0.5 \cdot 0.1 + 0.3 \cdot 0.0 + 0.9 \cdot 0.4 + 0.6 \cdot 0.1 = 0.47$$

Linear algebra reductions

Matrix multiplication. Given two N-by-N matrices, compute their product.

Brute force. N^3 flops.

problem	linear algebra	order of growth
matrix multiplication	$A \times B$	MM(N)
matrix inversion	A^{-1}	MM(N)
determinant	$\| A \|$	MM(N)
system of linear equations	$Ax = b$	MM(N)
LU decomposition	$A = L U$	MM(N)
least squares	$\min \|\|Ax - b\|\|_2$	MM(N)

numerical linear algebra problems with the same complexity as matrix multiplication

Q. Is brute-force algorithm optimal?

History of complexity of matrix multiplication

year	algorithm	order of growth
?	brute force	N^{3}
1969	Strassen	$N^{2.808}$
1978	Pan	$N^{2.796}$
1979	Bini	$N^{2.780}$
1981	Schönhage	$N^{2.522}$
1982	Romani	$N^{2.517}$
1982	Coppersmith-Winograd	$N^{2.496}$
1986	Strassen	$N^{2.479}$
1989	Coppersmith-Winograd	$N^{2.376}$
2010	Strother	$N^{2.3737}$
2011	Williams	$N^{2.3727}$
?	?	$N^{2+\varepsilon}$

number of floating-point operations to multiply two N-by-N matrices

Birds-eye view: review

Desiderata. Classify problems according to computational requirements.

complexity	order of growth	examples
linear	N	min, max, median, Burrows-Wheeler transform, …
linearithmic	$N \log N$	sorting, convex hull, closest pair, farthest pair, …
quadratic	N^2	?
⋮	⋮	⋮
exponential	c^N	?

Frustrating news. Huge number of problems have defied classification.

Birds-eye view: revised

Desiderata. Classify problems according to computational requirements.

complexity	order of growth	examples
linear	N	min, max, median, …
linearithmic	$N \log N$	sorting, convex hull, closest pair, farthest pair, …
$M(N)$	?	integer multiplication, division, square root, …
$MM(N)$	?	matrix multiplication, $Ax = b$, least square, determinant, …
⋮	⋮	⋮
NP-complete	probably not N^b	3-SAT, IND-SET, ILP, …

Good news. Can put many problems into equivalence classes.

Complexity zoo

Complexity class. Set of problems sharing some computational property.

https://complexityzoo.net/Complexity_Zoo

Bad news. Lots of complexity classes.

Summary

Reductions are important in theory to:

- Design algorithms.
- Establish lower bounds.
- Classify problems according to their computational requirements.

Reductions are important in practice to:

- Design algorithms.
- Design reusable software modules.
  - stacks, queues, priority queues, symbol tables, sets, graphs
  - sorting, regular expressions, Delaunay triangulation
  - MST, shortest path, maxflow, linear programming
- Determine difficulty of your problem and choose the right tool.
  - use exact algorithm for tractable problems
  - use heuristics for intractable problems

Advanced Topics

- Reductions

- Intractability
- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

Questions about computation

Q. What is a general-purpose computer?

Q. Are there limits on the power of digital computers?

Q. Are there limits on the power of machines we can build?

David Hilbert

Kurt Gödel

Alan Turing

Alonzo Church

John von Neumann

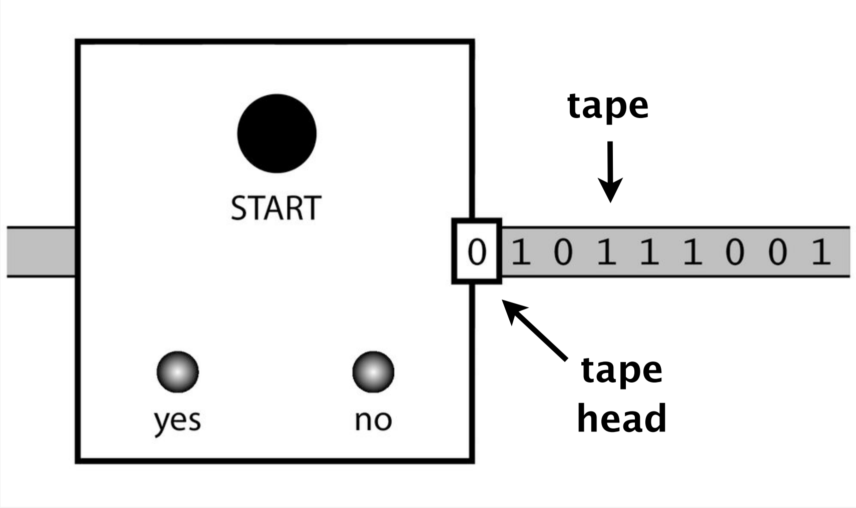
A simple model of computation: DFAs

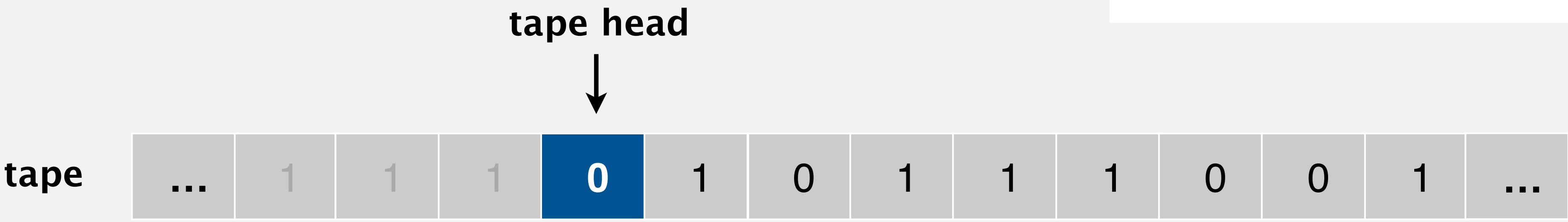
Tape.

- Stores input.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.

- Points to one cell of tape.
- Reads a symbol from active cell.
- Moves one cell at a time.





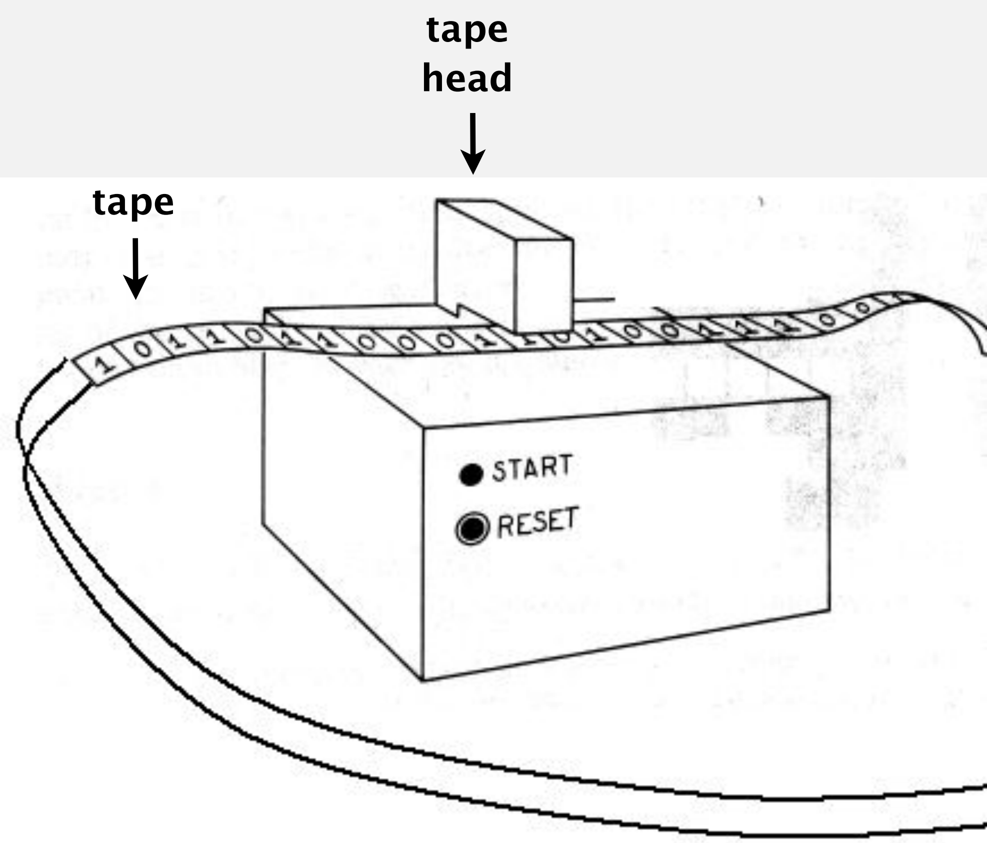
Q. Is there a more powerful model of computation?

A. Yes.

A universal model of computation: Turing machines

Tape.

- Stores input, output, and intermediate results.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.



Tape head.

- Points to one cell of tape.
- Reads a symbol from active cell.
- Writes a symbol to active cell.
- Moves one cell at a time.

tape head ↓

tape	…	#	1	1	**0**	0	+	1	0	1	1	#	…

Q. Is there a more powerful model of computation?

A. No! ⟵ most important scientific result of 20th century?

Church-Turing thesis (1936)

Turing machines can compute any function that can be computed by a physically harnessable process of the natural world.

Remark. "Thesis" and not a mathematical theorem because it's a statement about the physical world and not subject to proof.

but can be falsified

Use simulation to prove models equivalent.

- Android simulator on iPhone.
- iPhone simulator on Android.

Implications.

- No need to seek more powerful machines or languages.
- Enables rigorous study of computation (in this universe).

Bottom line. Turing machine is a simple and universal model of computation.

Church-Turing thesis: evidence

- 8 decades without a counterexample.
- Many, many models of computation that turned out to be equivalent. "universal"

model of computation	description
enhanced Turing machines	multiple heads, multiple tapes, 2D tape, nondeterminism
untyped lambda calculus	method to define and manipulate functions
recursive functions	functions dealing with computation on integers
unrestricted grammars	iterative string replacement rules used by linguists
extended L-systems	parallel string replacement rules that model plant growth
programming languages	Java, C, C++, Perl, Python, PHP, Lisp, PostScript, Excel
random access machines	registers plus main memory, e.g., TOY, Pentium
cellular automata	cells which change state based on local interactions
quantum computer	compute using superposition of quantum states
DNA computer	compute using biological operations on DNA

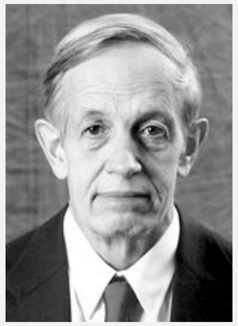
A question about algorithms

Q. Which algorithms are useful in practice?

- Measure running time as a function of input size N.
- Useful in practice ("efficient") = polynomial time for all inputs.

$a\,N^{\,b}$

von Neumann (1953)



Nash (1955)

Gödel (1956)

Cobham (1964)

Edmonds (1965)

Rabin (1966)

Ex 1. Sorting N items takes $N \log N$ compares using mergesort.

Ex 2. Finding best TSP tour on N points takes $N\,!$ steps using brute search.

Theory. Definition is broad and robust.

constants a and b tend to be small, e.g., $3\,N^{\,2}$

Practice. Poly-time algorithms scale to huge problems.

Exponential growth

Exponential growth dwarfs technological change.

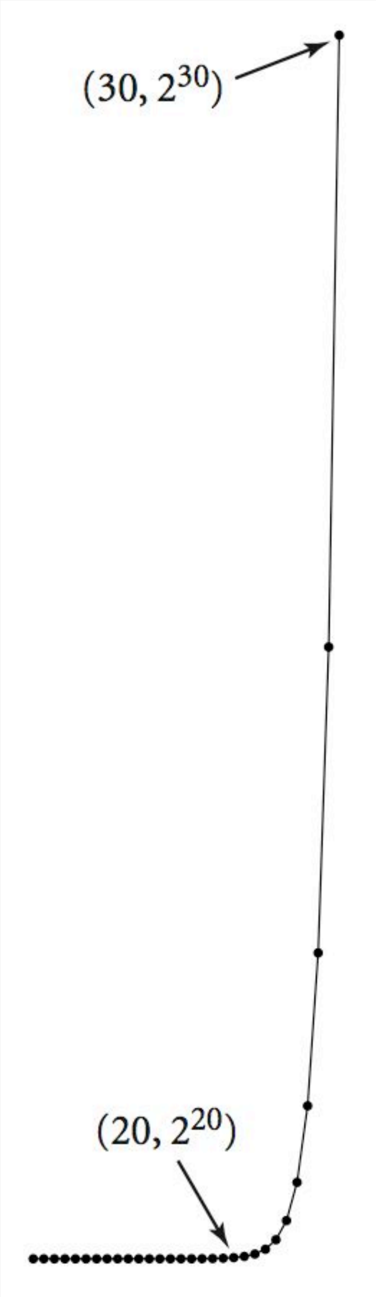
- Suppose you have a giant parallel computing device…
- With as many processors as electrons in the universe…
- And each processor has power of today's supercomputers…
- And each processor works for the life of the universe…

quantity	value
electrons in universe †	10^{79}
supercomputer instructions per second †	10^{13}
age of universe in seconds †	10^{17}

† estimated

- Will not help solve 1,000 city TSP problem via brute force.

$1000! >> 10^{1000} >> 10^{79} \times 10^{13} \times 10^{17}$



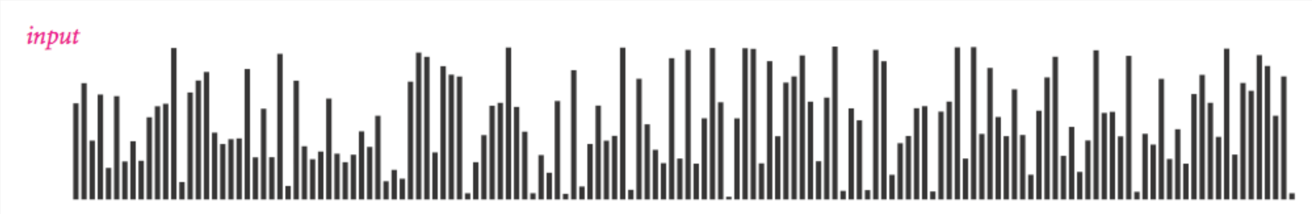
Questions about problems

Q. Which problems can we solve in practice?

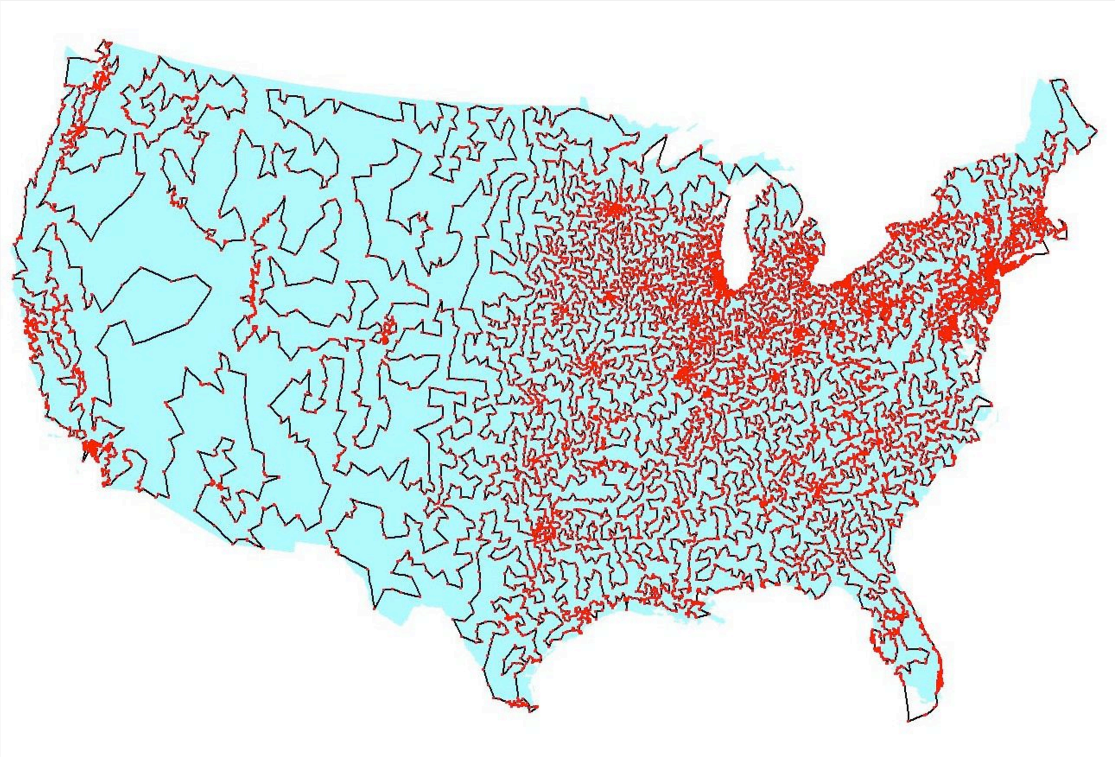
A. Those with poly-time algorithms.

Q. Which problems have poly-time algorithms?

A. Not so easy to know. Focus of today's lecture.



many known poly-time algorithms for sorting



no known poly-time algorithm for TSP

Bird's-eye view

Def. A problem is intractable if it can't be solved in polynomial time.

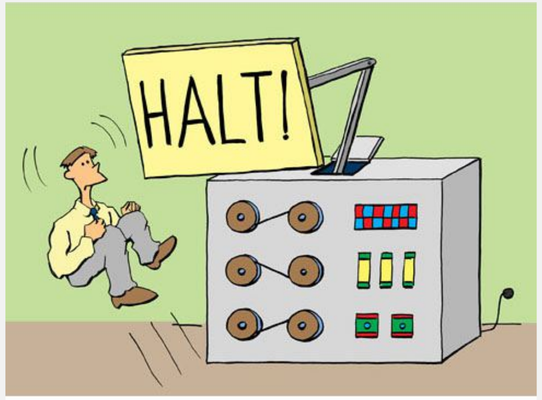
Desiderata. Prove that a problem is intractable.

input size = c + lg K

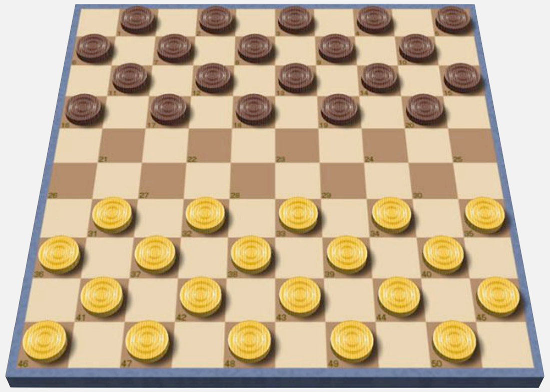
Two problems that provably require exponential time.

- Given a constant-size program, does it halt in at most K steps?
- Given N-by-N checkers board position, can the first player force a win?

using forced capture rule



Alan designed the perfect computer



Frustrating news. Very few successes.

Intractability

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

Four fundamental problems

LSOLVE. Given a system of linear equations, find a solution.

$$\begin{aligned} 0x_0 &+ 1x_1 &+ 1x_2 &= 4 \\ 2x_0 &+ 4x_1 &- 2x_2 &= 2 \\ 0x_0 &+ 3x_1 &+ 15x_2 &= 36 \end{aligned}$$

$$\begin{aligned} x_0 &= -1 \\ x_1 &= 2 \\ x_2 &= 2 \end{aligned}$$

← variables are real numbers

LP. Given a system of linear inequalities, find a solution.

$$\begin{aligned} 48x_0 &+ 16x_1 &+ 119x_2 &\le 88 \\ 5x_0 &+ 4x_1 &+ 35x_2 &\ge 13 \\ 15x_0 &+ 4x_1 &+ 20x_2 &\ge 23 \\ x_0 &, \; x_1 &, \; x_2 &\ge 0 \end{aligned}$$

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 1 \\ x_2 &= 1/5 \end{aligned}$$

← variables are real numbers

ILP. Given a system of linear inequalities, find a 0-1 solution.

$$\begin{aligned} & & x_1 &+ x_2 &\ge 1 \\ x_0 & & &+ x_2 &\ge 1 \\ x_0 &+ & x_1 &+ x_2 &\le 2 \end{aligned}$$

$$\begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ x_2 &= 1 \end{aligned}$$

← variables are 0 or 1

SAT. Given a system of boolean equations, find a binary solution.

$$\begin{aligned} (x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) &= \text{true} \\ (x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) &= \text{false} \\ (x_0 \text{ or } x_2) \text{ and } (x'_0) &= \text{true} \end{aligned}$$

$$\begin{aligned} x_0 &= \text{false} \\ x_1 &= \text{false} \\ x_2 &= \text{true} \end{aligned}$$

← variables are true or false

Four fundamental problems

LSOLVE. Given a system of linear equations, find a solution.

LP. Given a system of linear inequalities, find a solution.

ILP. Given a system of linear inequalities, find a 0-1 solution.

SAT. Given a system of boolean equations, find a binary solution.

Q. Which of these problems have poly-time algorithms?

- LSOLVE. Yes. Gaussian elimination solves N-by-N system in N^3 time.
- LP. Yes. Ellipsoid algorithm is poly-time. ← but was open problem for decades
- ILP, SAT. No poly-time algorithm known or believed to exist!

but we still don't know for sure

Search problems

Search problem. Given an instance I of a problem, find a solution S.

Requirement. Must be able to efficiently check that S is a solution.

poly-time in size of instance I

or report none exists

Search problems

Search problem. Given an instance I of a problem, find a solution S.

Requirement. Must be able to efficiently check that S is a solution.

LSOLVE. Given a system of linear equations, find a solution.

$$\begin{aligned} 0x_0 + 1x_1 + 1x_2 &= 4 \\ 2x_0 + 4x_1 - 2x_2 &= 2 \\ 0x_0 + 3x_1 + 15x_2 &= 36 \end{aligned}$$

instance I

$$\begin{aligned} x_0 &= -1 \\ x_1 &= 2 \\ x_2 &= 2 \end{aligned}$$

solution S

To check solution S, plug in values and verify each equation.

Search problems

Search problem. Given an instance I of a problem, find a solution S.

Requirement. Must be able to efficiently check that S is a solution.

LP. Given a system of linear inequalities, find a solution.

$$\begin{array}{rrrcl} 48x_0 & +16x_1 & +119x_2 & \leq & 88 \\ 5x_0 & +\ 4x_1 & +\ 35x_2 & \geq & 13 \\ 15x_0 & +\ 4x_1 & +\ 20x_2 & \geq & 23 \\ x_0 & ,\quad x_1 & ,\quad x_2 & \geq & 0 \end{array}$$

instance I

$$\begin{array}{rcl} x_0 & = & 1 \\ x_1 & = & 1 \\ x_2 & = & 1/5 \end{array}$$

solution S

To check solution S, plug in values and verify each inequality.

Search problems

Search problem. Given an instance I of a problem, find a solution S.

Requirement. Must be able to efficiently check that S is a solution.

ILP. Given a system of linear inequalities, find a binary solution.

$$\begin{array}{rcrcrcl} & & x_1 & + & x_2 & \geq & 1 \\ x_0 & & & + & x_2 & \geq & 1 \\ x_0 & + & x_1 & + & x_2 & \leq & 2 \end{array}$$

instance I

$$\begin{array}{rcl} x_0 & = & 0 \\ x_1 & = & 1 \\ x_2 & = & 1 \end{array}$$

solution S

To check solution S, plug in values and verify each inequality.

Search problems

Search problem. Given an instance I of a problem, find a solution S.

Requirement. Must be able to efficiently check that S is a solution.

SAT. Given a system of boolean equations, find a boolean solution.

$$
\begin{array}{llll}
(x'_1 \ or \ x'_2) & and & (x_0 \ or \ x_2) & = true \\
(x_0 \ or \ x_1) & and & (x_1 \ or \ x'_2) & = false \\
(x_0 \ or \ x_2) & and & (x'_0) & = true
\end{array}
$$

instance I

$$
\begin{array}{l}
x_0 = false \\
x_1 = false \\
x_2 = true
\end{array}
$$

solution S

To check solution S, plug in values and verify each equation.

Search problems

Search problem. Given an instance I of a problem, find a solution S.

Requirement. Must be able to efficiently check that S is a solution.

FACTOR. Given an n-bit integer x, find a nontrivial factor.

input size = number of bits

147573952589676412927	193707721
instance I	**solution S**

To check solution S, long divide 193707721 into 147573952589676412927.

Intractability

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

NP

Def. NP is the class of all search problems.

Note: classic definition limits NP to yes–no problems

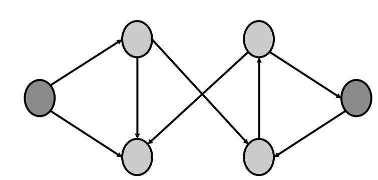
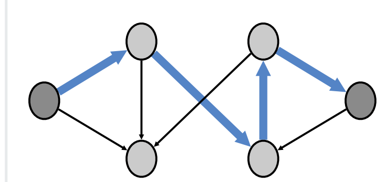
problem	description	poly-time algorithm	instance I	solution S
LSOLVE (A, b)	Find a vector x that satisfies $Ax = b$	Gaussian elimination	$\begin{array}{rrrcr} 0x_0 & + 1x_1 & + 1x_2 & = & 4 \\ 2x_0 & + 4x_1 & - 2x_2 & = & 2 \\ 0x_0 & + 3x_1 & + 15x_2 & = & 36 \end{array}$	$\begin{array}{lcr} x_0 & = & -1 \\ x_1 & = & 2 \\ x_2 & = & 2 \end{array}$
LP (A, b)	Find a vector x that satisfies $Ax \le b$	ellipsoid	$\begin{array}{rrrcr} 48x_0 & + 16x_1 & + 119x_2 & \le & 88 \\ 5x_0 & + 4x_1 & + 35x_2 & \ge & 13 \\ 15x_0 & + 4x_1 & + 20x_2 & \ge & 23 \\ x_0 & , x_1 & , x_2 & \ge & 0 \end{array}$	$\begin{array}{lcr} x_0 & = & 1 \\ x_1 & = & 1 \\ x_2 & = & 1/5 \end{array}$
ILP (A, b)	Find a binary vector x that satisfies $Ax \le b$	???	$\begin{array}{rrrcr} & x_1 & + x_2 & \ge & 1 \\ x_0 & & + x_2 & \ge & 1 \\ x_0 & + x_1 & + x_2 & \le & 2 \end{array}$	$\begin{array}{lcr} x_0 & = & 0 \\ x_1 & = & 1 \\ x_2 & = & 1 \end{array}$
SAT (Φ, b)	Find a boolean vector x that satisfies $\Phi(x) = b$	???	$(x'_1 \text{ or } x'_2)$ *and* $(x_0 \text{ or } x_2) = \textit{true}$ $(x_0 \text{ or } x_1)$ *and* $(x_1 \text{ or } x'_2) = \textit{false}$ $(x_0 \text{ or } x_2)$ *and* $(x'_0) = \textit{true}$	$x_0 = \textit{false}$ $x_1 = \textit{false}$ $x_2 = \textit{true}$
FACTOR (x)	Find a nontrivial factor of the integer x	???	147573952589676412927	193707721

Significance. What scientists and engineers aspire to compute feasibly.

P

Def. P is the class of search problems solvable in poly-time.

Note: classic definition limits P to yes–no problems

problem	description	poly-time algorithm	instance I	solution S
LSOLVE (A, b)	Find a vector x that satisfies $Ax = b$	Gaussian elimination (Edmonds 1967)	$\begin{array}{rrrcr} 0x_0 & + 1x_1 & + 1x_2 & = & 4 \\ 2x_0 & + 4x_1 & - 2x_2 & = & 2 \\ 0x_0 & + 3x_1 & + 15x_2 & = & 36 \end{array}$	$\begin{array}{rcr} x_0 & = & -1 \\ x_1 & = & 2 \\ x_2 & = & 2 \end{array}$
LP (A, b)	Find a vector x that satisfies $Ax \le b$	ellipsoid (Khachiyan 1979)	$\begin{array}{rrrcr} 48x_0 & + 16x_1 & + 119x_2 & \le & 88 \\ 5x_0 & + 4x_1 & + 35x_2 & \ge & 13 \\ 15x_0 & + 4x_1 & + 20x_2 & \ge & 23 \\ x_0 & , \; x_1 & , \; x_2 & \ge & 0 \end{array}$	$\begin{array}{rcr} x_0 & = & 1 \\ x_1 & = & 1 \\ x_2 & = & \frac{1}{5} \end{array}$
SORT (a)	Find a permutation that puts array a in order	mergesort (von Neumann 1945)	2.3 8.5 1.2 9.1 2.2 0.3	5 2 4 0 1 3
STCONN (G, s, t)	Find a path in a graph G from s to t	depth-first search (Theseus)		

Significance. What scientists and engineers do compute feasibly.

Nondeterminism

Nondeterministic machine can guess the desired solution.

recall NFA implementation

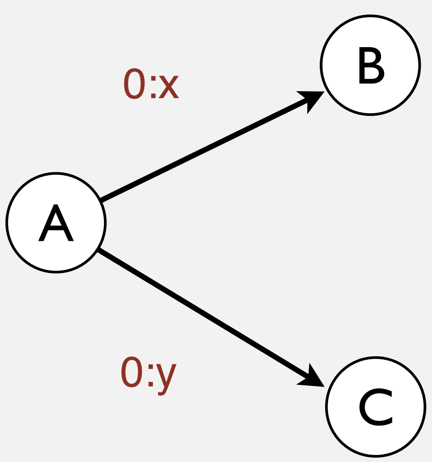
Ex. int[] a = new int[N];

- Java: initializes entries to 0.
- Nondeterministic machine: initializes entries to the solution!

ILP. Given a system of linear inequalities, guess a 0-1 solution.

$$\begin{array}{ccccccc} & & x_1 & + & x_2 & \geq & 1 \\ x_0 & & & + & x_2 & \geq & 1 \\ x_0 & + & x_1 & + & x_2 & \leq & 2 \end{array}$$

$$\begin{array}{ccc} x_0 & = & 0 \\ x_1 & = & 1 \\ x_2 & = & 1 \end{array}$$



Ex. Turing machine.

- Deterministic: state, input determines next state.
- Nondeterministic: more than one possible next state.

NP. Search problems solvable in poly time on a nondeterministic TM.

Extended Church-Turing thesis

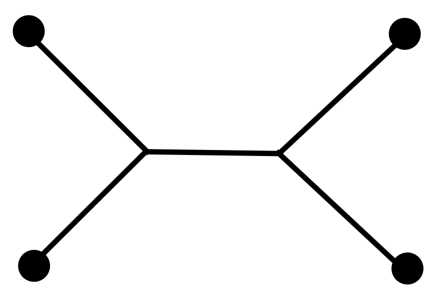
P = search problems solvable in poly-time in the natural world.

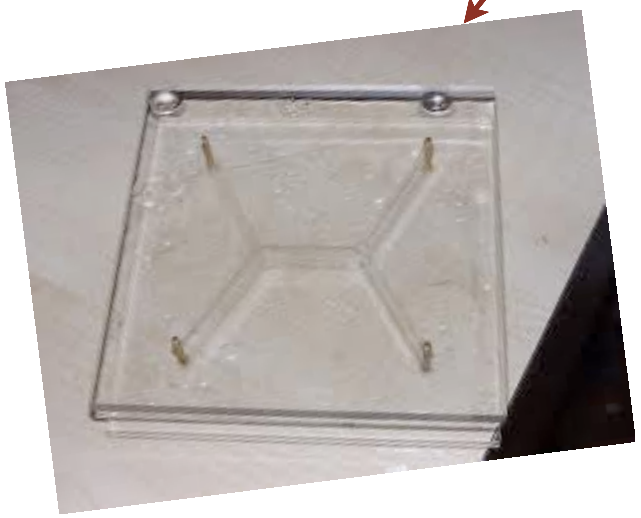
Evidence supporting thesis. True for all physical computers.

Natural computers? No successful attempts (yet).

doesn't work
for large N

- Ex. Computing Steiner trees with soap bubbles





- STEINER: Find set of lines of minimal length
- connecting N given points

Implication. To make future computers more efficient,
suffices to focus on improving implementation of existing designs.

P vs. NP

Does P = NP ?

Copyright © 1990, Matt Groening

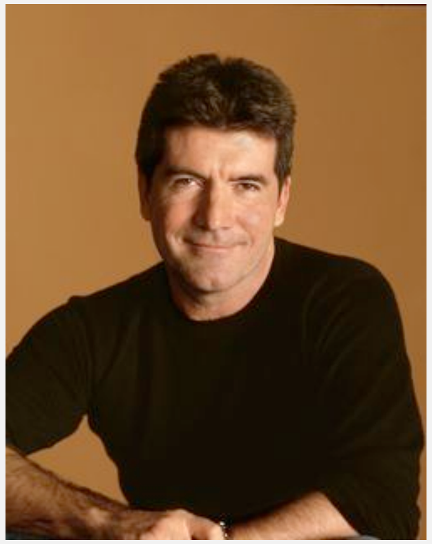
Copyright © 2000, Twentieth Century Fox

Automating creativity

Q. Being creative vs. appreciating creativity?

Ex. Mozart composes a piece of music; our neurons appreciate it.
Ex. Wiles proves a deep theorem; a colleague referees it.
Ex. Boeing designs an efficient airfoil; a simulator verifies it.
Ex. Einstein proposes a theory; an experimentalist validates it.

creative



ordinary

Computational analog. Does P = NP?

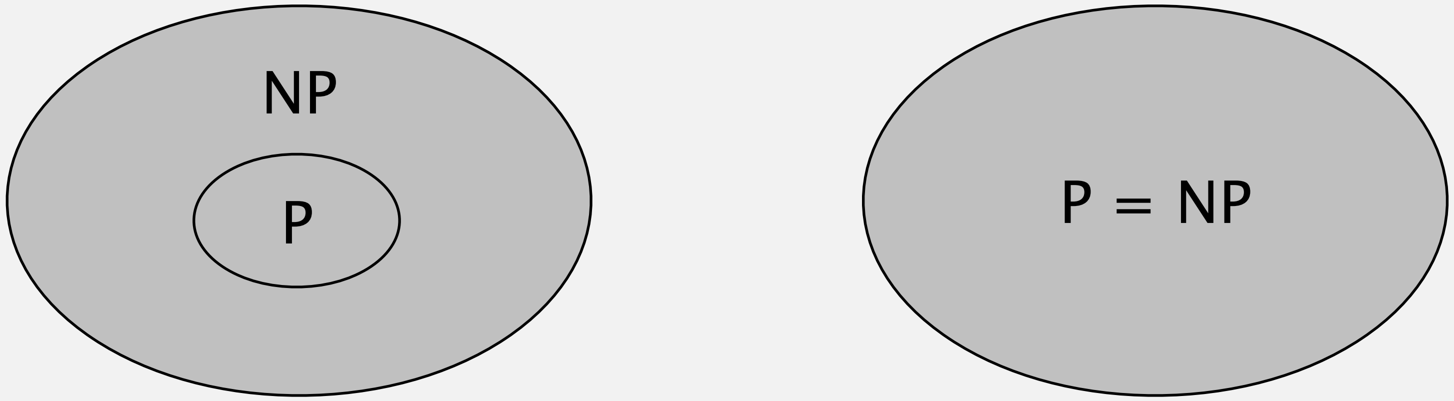
The central question

P. Class of search problems solvable in poly-time.

NP. Class of all search problems.

Does P = NP ? [Can you always avoid brute-force searching and do better]

Two worlds.



P ≠ NP   P = NP

If P = NP… Poly-time algorithms for SAT, ILP, TSP, FACTOR, …

If P ≠ NP… Would learn something fundamental about our universe.

Overwhelming consensus. P ≠ NP.

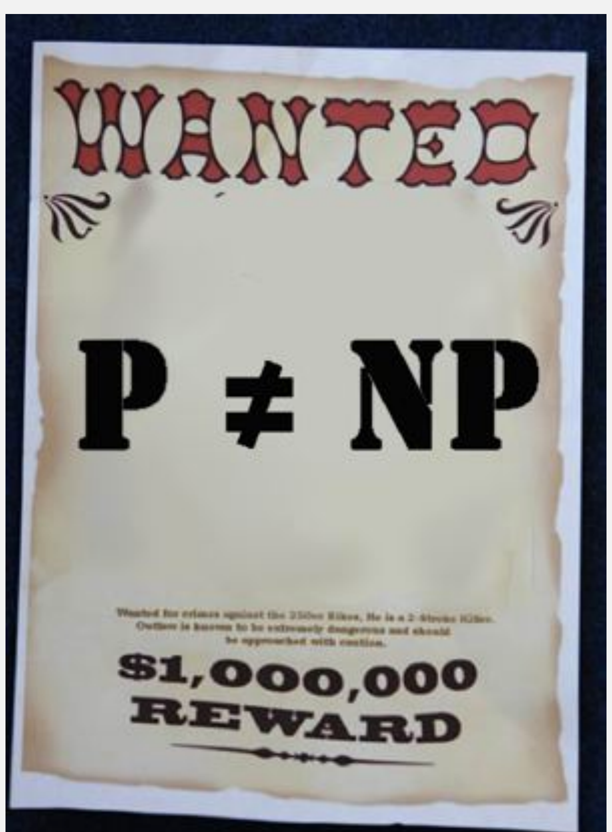
The central question

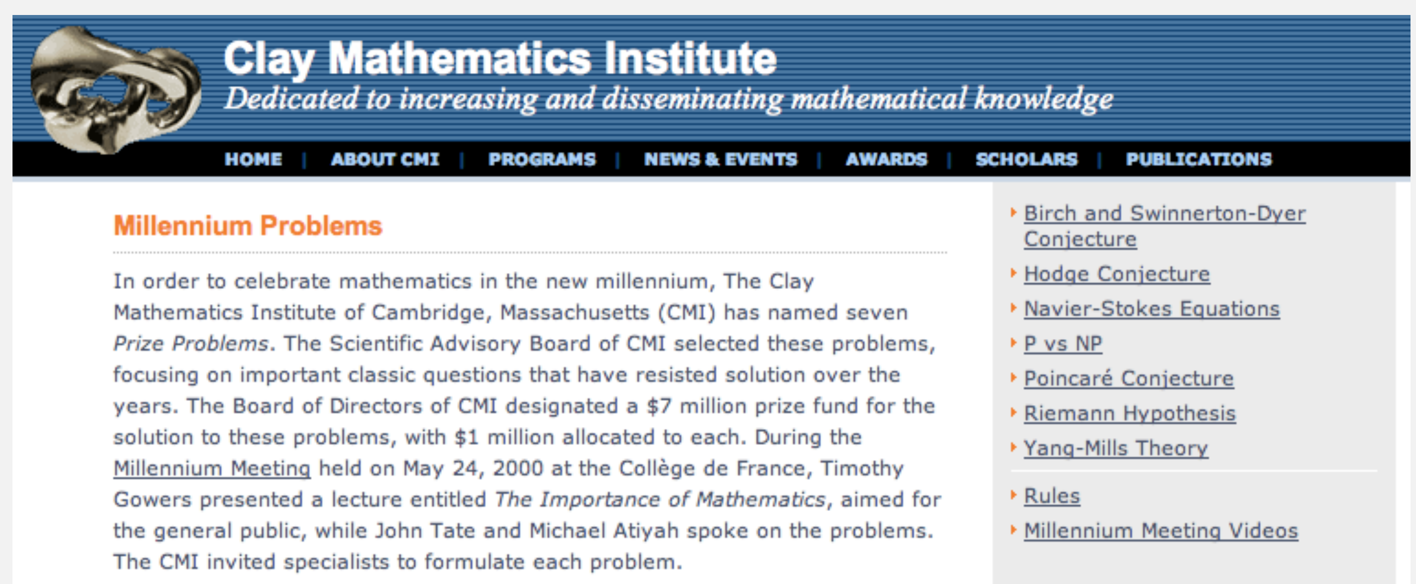
P. Class of search problems solvable in poly-time.

NP. Class of all search problems.

Does P = NP ? [Can you always avoid brute-force searching and do better]

Millennium prize. $1 million for resolution of P = NP problem.





Intractability

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

A key problem: satisfiability

SAT. Given a system of boolean equations, find a solution.

$$
\begin{aligned}
x'_1 \text{ or } x_2 \text{ or } x_3 &= \text{true} \\
x_1 \text{ or } x'_2 \text{ or } x_3 &= \text{true} \\
x'_1 \text{ or } x'_2 \text{ or } x'_3 &= \text{true} \\
x'_1 \text{ or } x'_2 \text{ or } x_4 &= \text{true}
\end{aligned}
$$

Key applications.

- Automatic verification systems for software.
- Electronic design automation (EDA) for hardware.
- Mean field diluted spin glass model in physics.
- ...

Exhaustive search

Q. How to solve an instance of SAT with n variables?

A. Exhaustive search: try all 2^n truth assignments.

Q. Can we do anything substantially more clever?

Conjecture. $\underbrace{\text{No poly-time algorithm}}_{\text{"intractable"}}$ for SAT.

Classifying problems

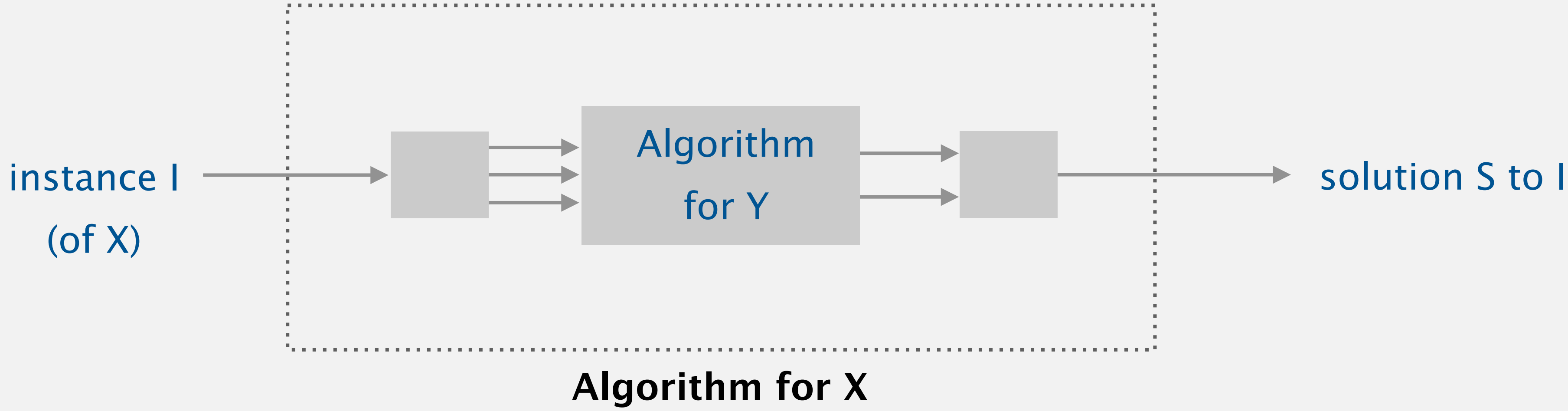
Q. Which search problems are in P?

A. No easy answers (we don't even know whether P = NP).

Cook reduction

Problem X poly-time reduces to problem Y if X can be solved with:

- Polynomial number of standard computational steps.
- Polynomial number of calls to Y.



Consequence. If SAT poly-time reduces to Y, then we conclude that Y is (probably) intractable.

SAT poly-time reduces to ILP

SAT. Given a system of boolean equations, find a solution.

$$
\begin{aligned}
x'_1 \text{ or } x_2 \text{ or } x_3 &= \textit{true} \\
x_1 \text{ or } x'_2 \text{ or } x_3 &= \textit{true} \\
x'_1 \text{ or } x'_2 \text{ or } x'_3 &= \textit{true} \\
x'_1 \text{ or } x'_2 \text{ or } x_4 &= \textit{true}
\end{aligned}
$$

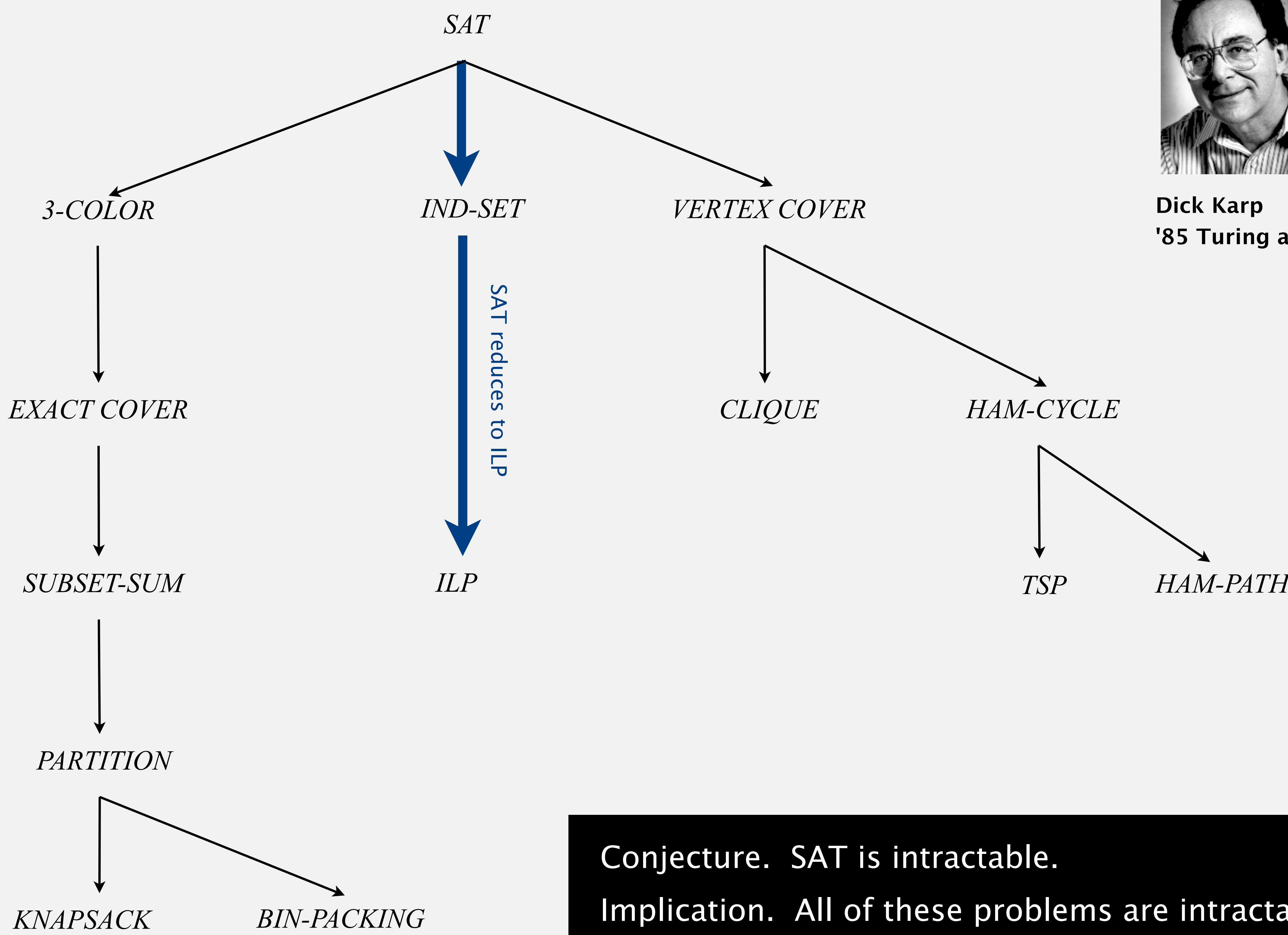
can to reduce any SAT problem to this form

ILP. Given a system of linear inequalities, find a 0-1 solution.

$$
\begin{aligned}
1 &\leq (1 - x_1) + x_2 + x_3 \\
1 &\leq x_1 + (1 - x_2) + x_3 \\
1 &\leq (1 - x_1) + (1 - x_2) + (1 - x_3) \\
1 &\leq (1 - x_1) + (1 - x_2) + x_4
\end{aligned}
$$

solution to this ILP instance gives solution to original SAT instance

More poly-time reductions from boolean satisfiability



Dick Karp
'85 Turing award

Conjecture. SAT is intractable.

Implication. All of these problems are intractable.

Still more reductions from SAT

Aerospace engineering. Optimal mesh partitioning for finite elements.

Biology. Phylogeny reconstruction.

Chemical engineering. Heat exchanger network synthesis.

Chemistry. Protein folding.

Civil engineering. Equilibrium of urban traffic flow.

Economics. Computation of arbitrage in financial markets with friction.

Electrical engineering. VLSI layout.

Environmental engineering. Optimal placement of contaminant sensors.

Financial engineering. Minimum risk portfolio of given return.

Game theory. Nash equilibrium that maximizes social welfare.

Mathematics. Given integer $a_1, \ldots, a_n$, compute $\int_0^{2\pi} \cos(a_1\theta) \times \cos(a_2\theta) \times \cdots \times \cos(a_n\theta)\, d\theta$

Mechanical engineering. Structure of turbulence in sheared flows.

Medicine. Reconstructing 3d shape from biplane angiocardiogram.

Operations research. Traveling salesperson problem.

Physics. Partition function of 3d Ising model.

Politics. Shapley-Shubik voting power.

Recreation. Versions of Sudoko, Checkers, Minesweeper, Tetris.

Statistics. Optimal experimental design.

plus over 6,000 scientific papers per year

Intractability

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

NP-completeness

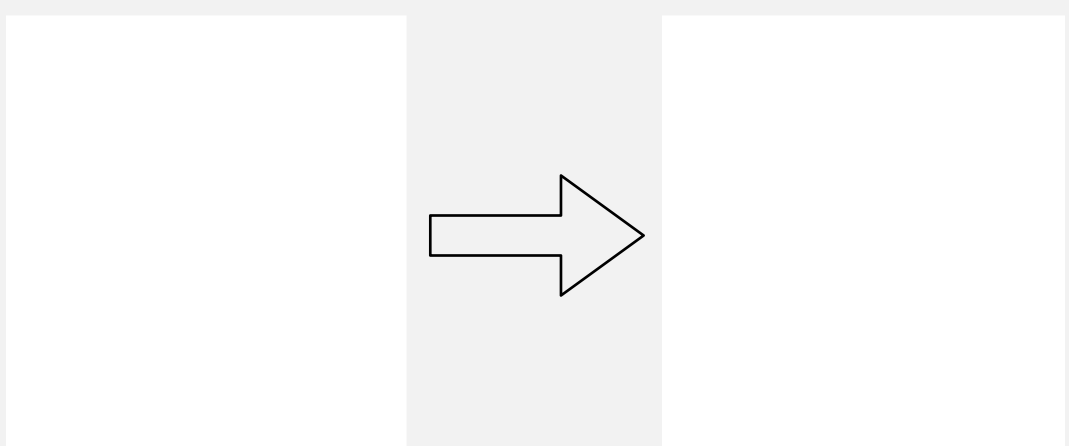
Def. An NP problem is NP-complete if every problem in NP poly-time reduce to it.

Proposition. [Cook 1971, Levin 1973] SAT is NP-complete.

every NP problem is a SAT problem in disguise

Extremely brief proof sketch:

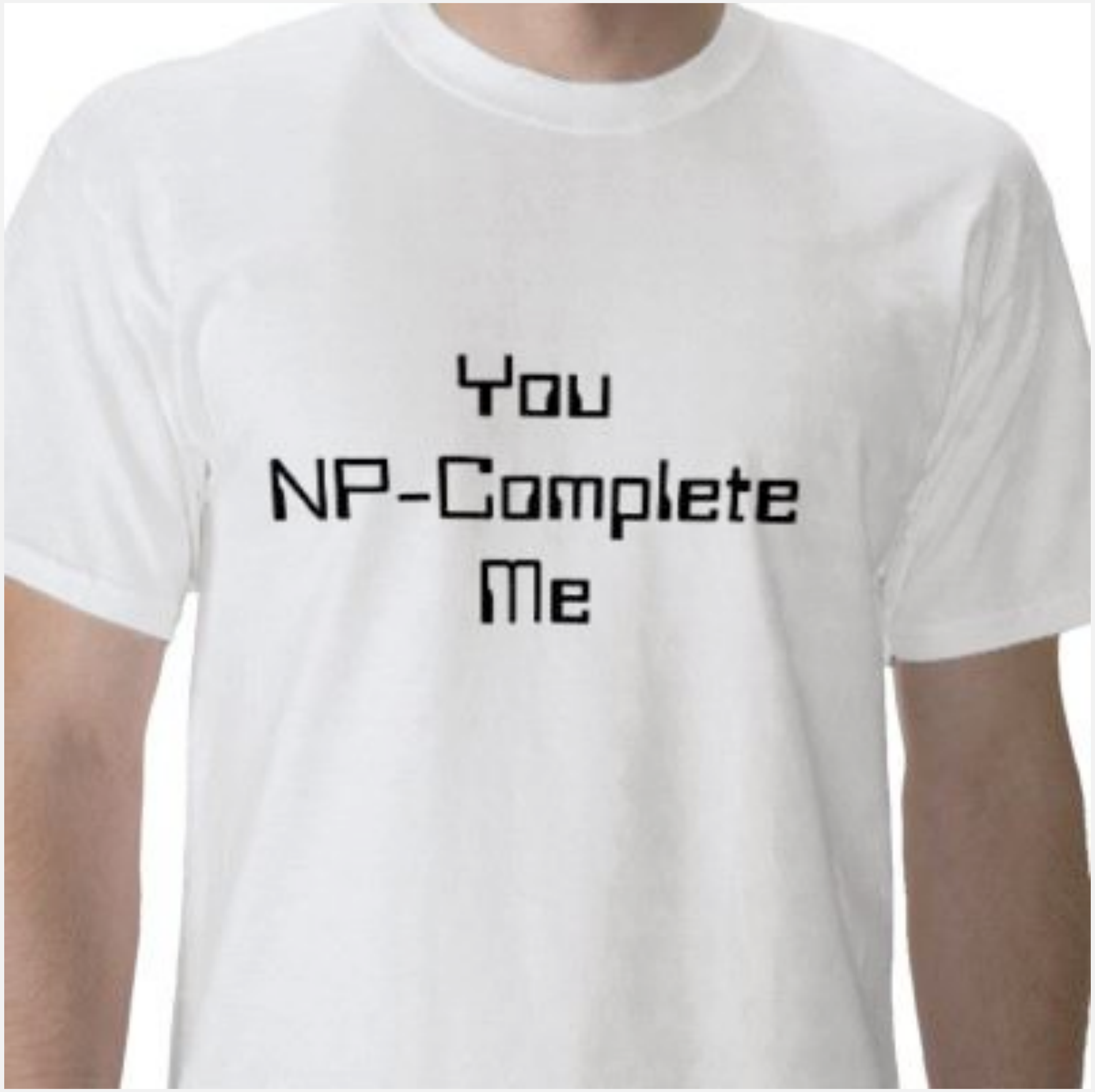
- Convert non-deterministic TM notation to SAT notation.
- If you can solve SAT, you can solve any problem in NP.



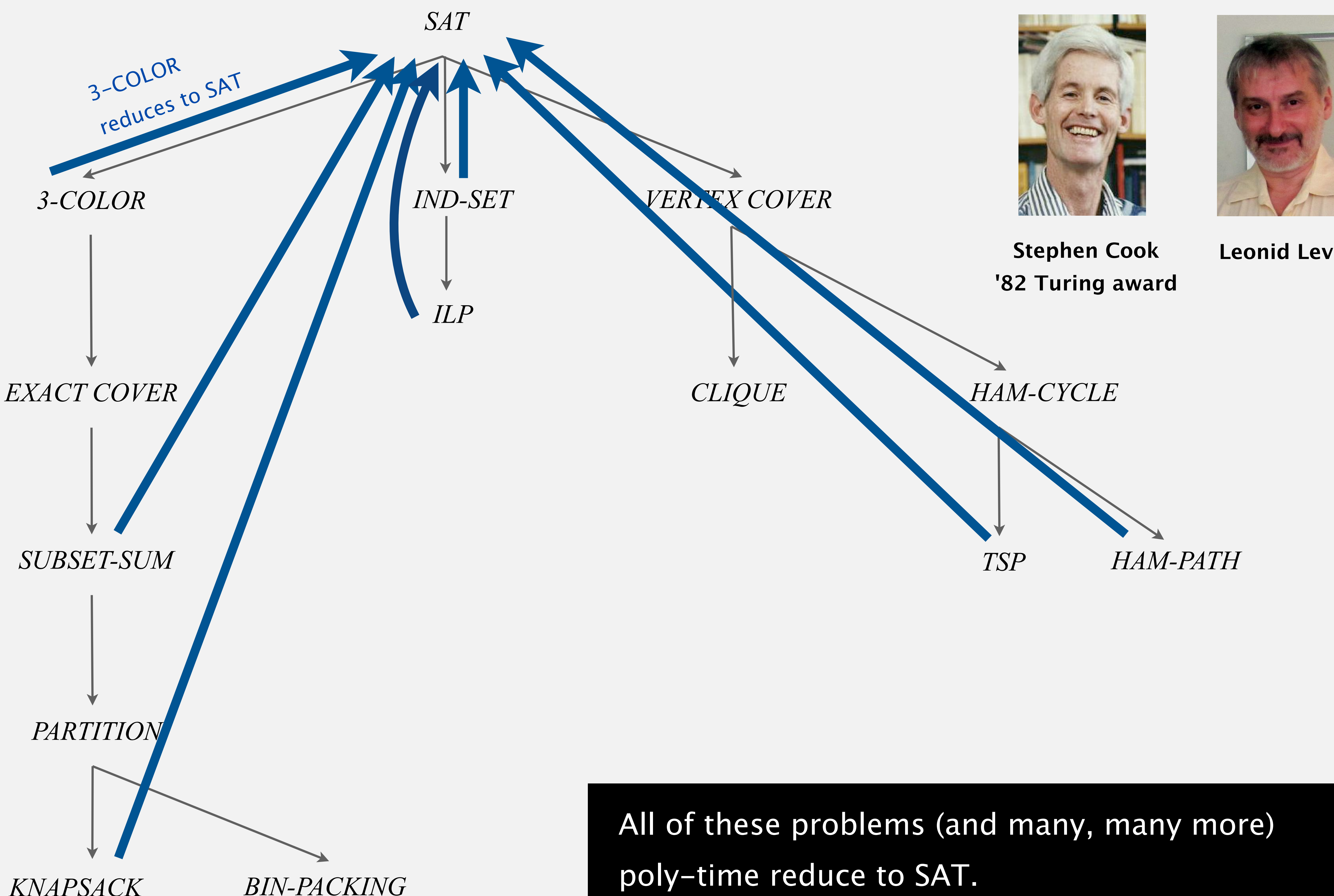
nondeterministic TM SAT instance

Corollary. Poly-time algorithm for SAT iff P = NP.

You NP-complete me



Implications of Cook-Levin theorem

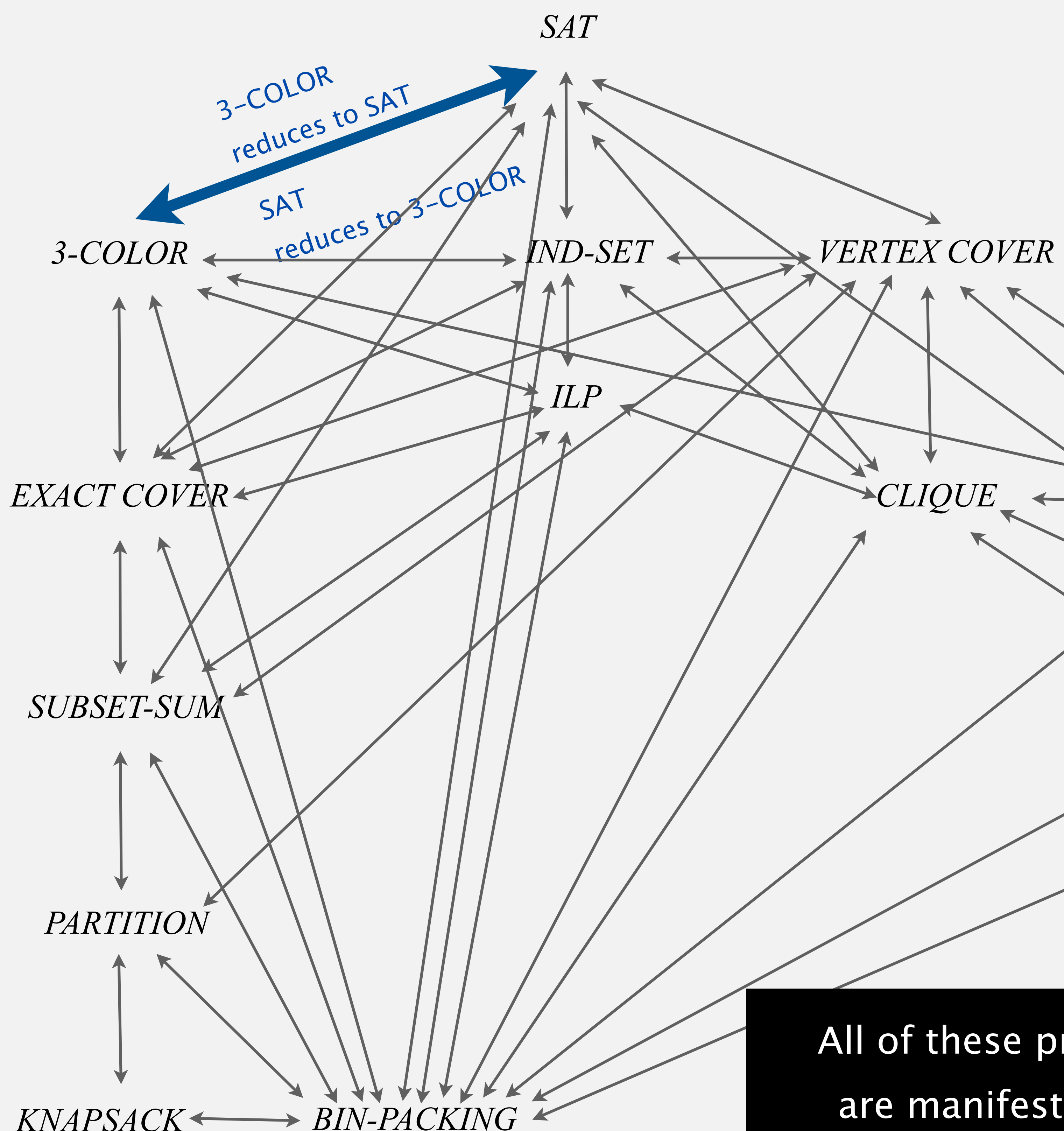


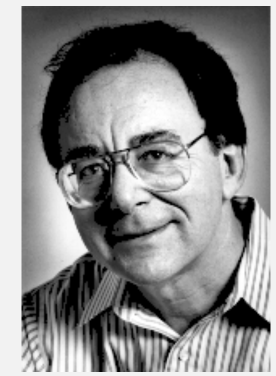
Stephen Cook
'82 Turing award

Leonid Levin

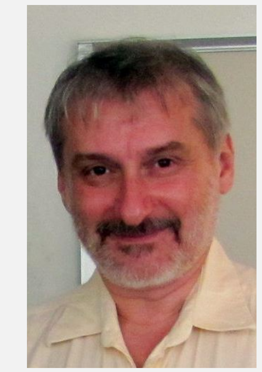
All of these problems (and many, many more) poly-time reduce to SAT.

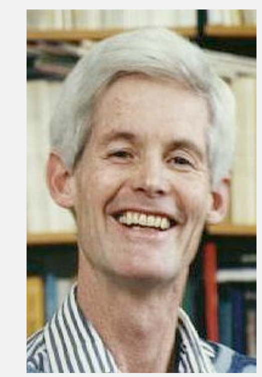
Implications of Karp + Cook-Levin





+





All of these problems are NP-complete; they are manifestations of the same really hard problem.

Implications of NP-Completeness

Implication. [SAT captures difficulty of whole class NP]

- Poly-time algorithm for SAT iff P = NP.
- No poly-time algorithm for some NP problem $\Rightarrow$ none for SAT.

Remark. Can replace SAT with any of Karp's problems.

Proving a problem NP-complete guides scientific inquiry.

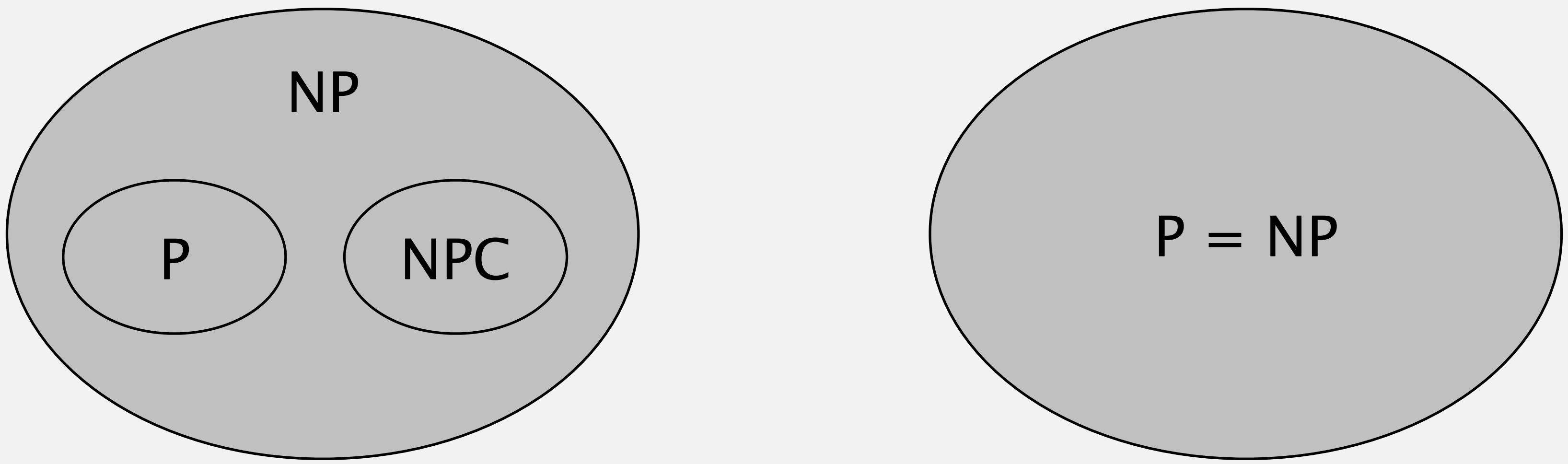
- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager finds closed form solution to 2D version in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: 3D-ISING proved NP-complete.

a holy grail of statistical mechanics

search for closed formula appears doomed

Two worlds (more detail)

Overwhelming consensus (still). $P \neq NP$.



P ≠ NP

P = NP

Why we believe $P \neq NP$.

> *" We admire Wiles' proof of Fermat's last theorem, the scientific theories of Newton, Einstein, Darwin, Watson and Crick, the design of the Golden Gate bridge and the Pyramids, precisely because they seem to require a leap which cannot be made by everyone, let alone a by simple mechanical device. "* — *Avi Wigderson*

Summary

P. Class of search problems solvable in poly-time.

NP. Class of all search problems, some of which seem wickedly hard.

NP-complete. Hardest problems in NP.

Intractable. Problem with no poly-time algorithm.

Many fundamental problems are NP-complete.

- SAT, ILP, HAMILTON-PATH, …
- 3D-ISING, …

Use theory a guide:

- A poly-time algorithm for an NP-complete problem would be a stunning breakthrough (a proof that P = NP).
- You will confront NP-complete problems in your career.
- Safe to assume that P ≠ NP and that such problems are intractable.
- Identify these situations and proceed accordingly.

Exploiting intractability

Modern cryptography.

- Ex. Send your credit card to Amazon.
- Ex. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

RSA cryptosystem.

- To use: multiply two n-bit integers. [poly-time]
- To break: factor a $2\,n$-bit integer. [unlikely poly-time]

Multiply = EASY

$23 \times 67 \longleftrightarrow 1{,}541$

Factor = HARD

Exploiting intractability

Challenge. Factor this number.

74037563479561712828046796097429573142593188889231289084936232638

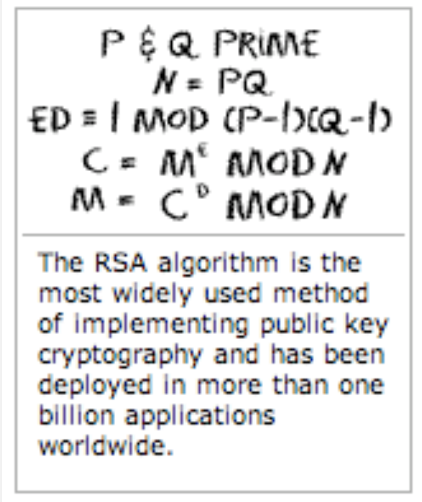
97276503402826627689199641962511784399589433050212758537011896809

82867331732731089309005525051168770632990723963807867100860969625

37934650563796359

RSA-704
($30,000 prize if you can factor)

Can't do it? Create a company based on the difficulty of factoring.



RSA algorithm

RSA sold for $2.1 billion

or design a t-shirt

Exploiting intractability

FACTOR. Given an n-bit integer x, find a nontrivial factor.

Q. What is complexity of FACTOR?

A. In NP, but not known (or believed) to be in P or NP-complete.

Q. What if P = NP?

A. Poly-time algorithm for factoring; modern e-conomy collapses.

Proposition. [Shor 1994] Can factor an n-bit integer in n^3 steps on a "quantum computer."

Q. Do we still believe the extended Church-Turing thesis???

Coping with intractability

Relax one of desired features.

- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

Special cases may be tractable.

- Ex: Linear time algorithm for 2-SAT. ← at most two variables per equation
- Ex: Linear time algorithm for Horn-SAT. ← at most one un-negated variable per equation

Coping with intractability

Relax one of desired features.

- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

Develop a heuristic, and hope it produces a good solution.

- No guarantees on quality of solution.
- Ex. TSP assignment heuristics.
- Ex. Metropolis algorithm, simulating annealing, genetic algorithms.

Approximation algorithm. Find solution of provably good quality.

- Ex. MAX-3SAT: provably satisfy 87.5% as many clauses as possible.

but if you can guarantee to satisfy 87.51% as many clauses
as possible in poly-time, then P = NP !

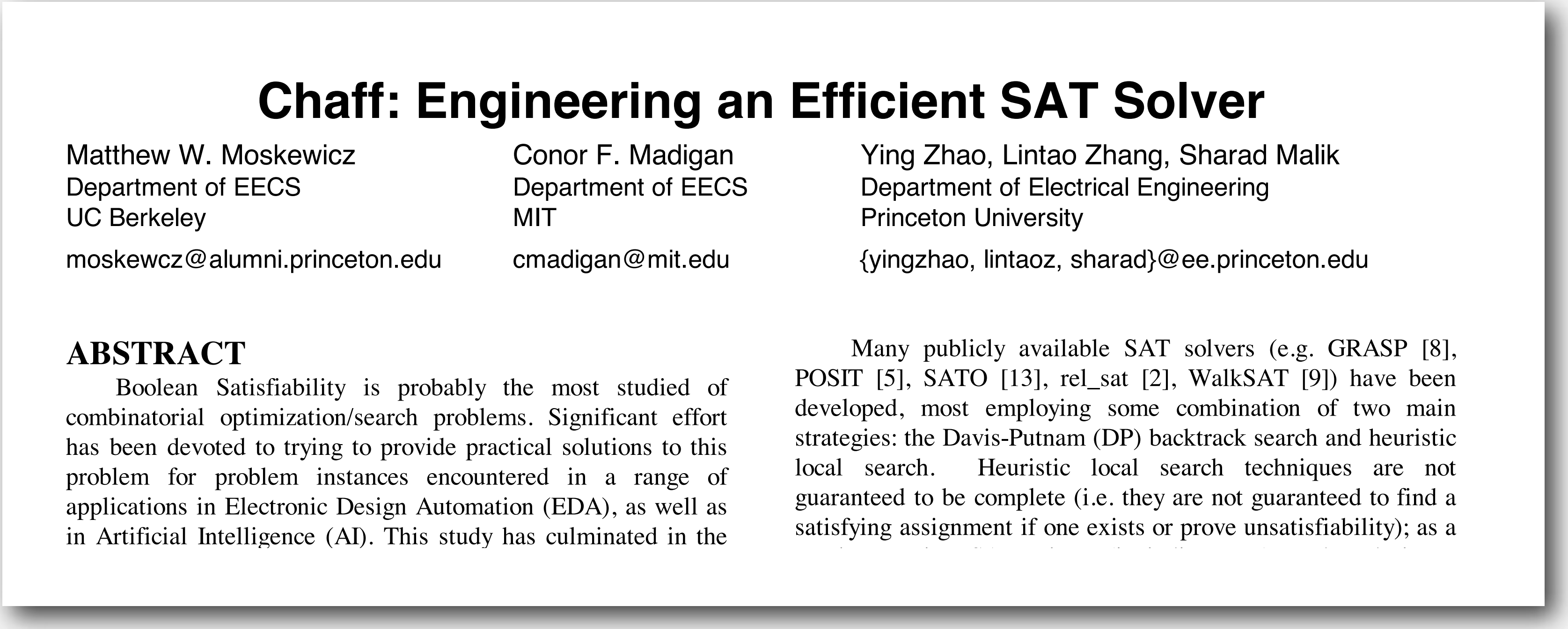
Coping with intractability

Relax one of desired features.

- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

Complexity theory deals with worst case behavior.

- Instance(s) you want to solve may be "easy."
- Chaff solves real-world SAT instances with ~ 10K variable.



Chaff: Engineering an Efficient SAT Solver

Matthew W. Moskewicz
Department of EECS
UC Berkeley
moskewcz@alumni.princeton.edu

Conor F. Madigan
Department of EECS
MIT
cmadigan@mit.edu

Ying Zhao, Lintao Zhang, Sharad Malik
Department of Electrical Engineering
Princeton University
{yingzhao, lintaoz, sharad}@ee.princeton.edu

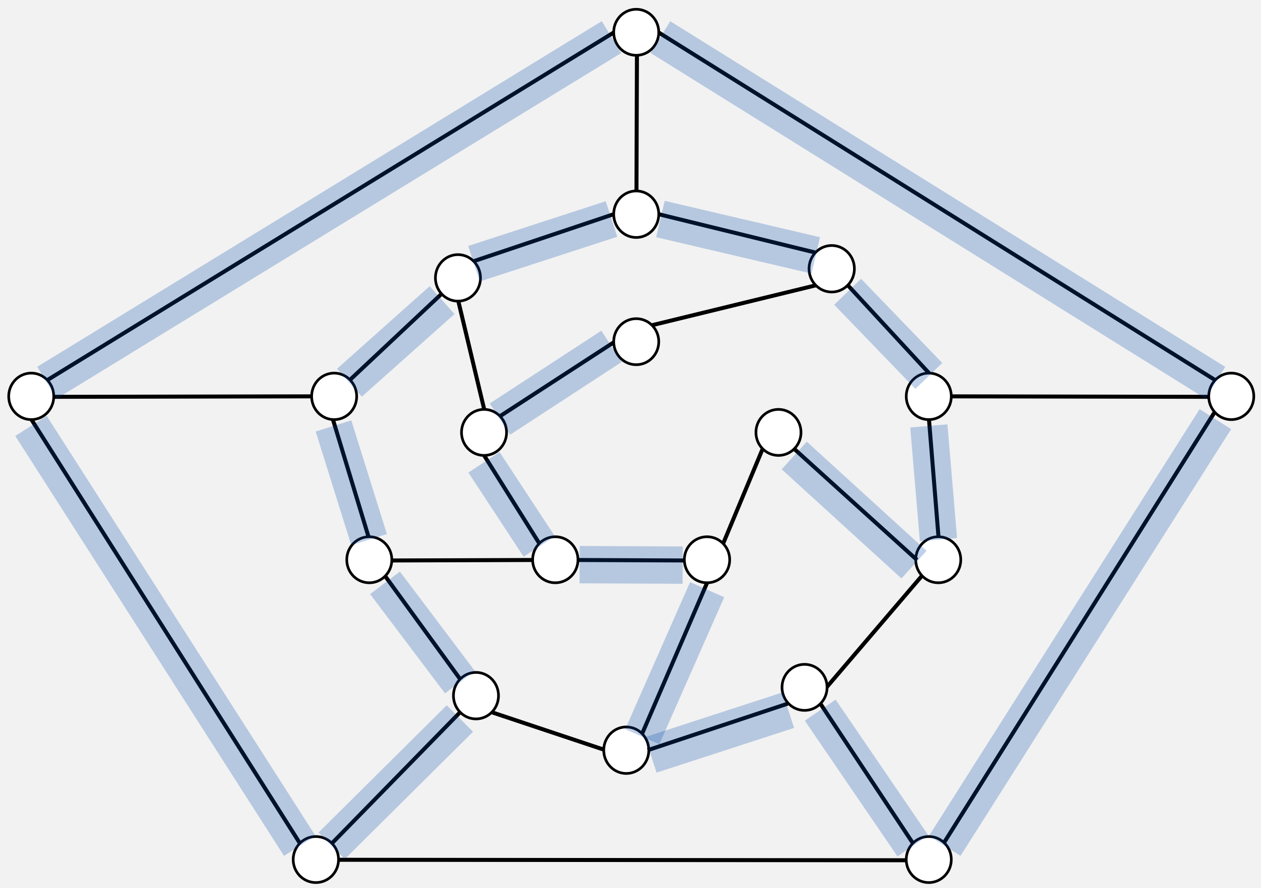
ABSTRACT

Boolean Satisfiability is probably the most studied of combinatorial optimization/search problems. Significant effort has been devoted to trying to provide practical solutions to this problem for problem instances encountered in a range of applications in Electronic Design Automation (EDA), as well as in Artificial Intelligence (AI). This study has culminated in the

Many publicly available SAT solvers (e.g. GRASP [8], POSIT [5], SATO [13], rel_sat [2], WalkSAT [9]) have been developed, most employing some combination of two main strategies: the Davis-Putnam (DP) backtrack search and heuristic local search. Heuristic local search techniques are not guaranteed to be complete (i.e. they are not guaranteed to find a satisfying assignment if one exists or prove unsatisfiability); as a

Hamilton path

Goal. Find a simple path that visits every vertex exactly once.



visit every edge exactly once

Remark. Euler path easy, but Hamilton path is NP-complete.

Hamilton path: Java implementation

```
public class HamiltonPath
{
   private boolean[] marked;     // vertices on current path
   private int count = 0;        // number of Hamiltonian paths

   public HamiltonPath(Graph G)
   {
      marked = new boolean[G.V()];
      for (int v = 0; v < G.V(); v++)
         dfs(G, v, 1);
   }

   private void dfs(Graph G, int v, int depth)
   {
      marked[v] = true;
      if (depth == G.V()) count++;

      for (int w : G.adj(v))
         if (!marked[w]) dfs(G, w, depth+1);

      marked[v] = false;
  }
}
```

found one

length of current path (depth of recursion)

backtrack if w is already part of path

clean up

That’s all, folks: keep searching!

The world’s longest path (Sendero de Chile): 9,700 km.
(originally scheduled for completion in 2010; now delayed until 2038)