P and NP

CSE 332 25Su
Lecture 23,24
*Credit to Robbie Weber for slides

* not testable material

Announcements

- EX11 Due Wednesday
- EX12 Released Today, due Friday
  - Note: No late days or extensions on EX12
  - We made it shorter than previous quarters
- Last call for Exam 2 makeups
  - https://courses.cs.washington.edu/courses/cse332/25su/exams/final.html
  - **Note: it will be hard to accommodate makeups; only four days to grade**
  - If you can't make proposed makeup dates (e.g., sickness/emergency), some options:
  - Option 1: Exam 1 is worth 40% instead of 20% of overall grade
  - Option 2: Take the final exam in the next CSE 332 offering

P vs. NP

Definition Dump

Efficient

We'll consider a problem "efficiently solvable" if it has a polynomial time algorithm.

I.e. an algorithm that runs in time $O(n^k)$ where k is a constant.

Are these algorithms always actually efficient?

Well………no

Your n^{10000} algorithm or even your $2^{2^{2^{2^2}}} \cdot n^3$ algorithm probably aren't going to finish anytime soon.

But these edge cases are rare, and polynomial time is good as a low bar

- If we can't even find an n^{10000} algorithm, we're probably not getting one that is efficient in practice anyway.

Some definitions

A problem is a set of inputs and the correct outputs.

"Find a Minimum Spanning Tree" is a problem.

- Input is a graph, output is the MST.

- "Tell whether a list is sorted" is a problem.
- Input is an array, output is "yes" or "no"

- "Sort this array" is a problem.
- Input is an array, output is the same numbers, now in sorted order.

Some definitions

An instance is a single input to a problem.

A single, particular graph is an instance of the MST problem

- A single, particular graph with vertices s and t is an instance of the Shortest Path problem.

- A single, particular array is an instance of the “is the array sorted?” problem.

13	24	31	30	35	39	51

Not Sorted

Decision Problems

Let’s go back to dividing problems into solvable/not solvable.
For today, we’re going to talk about **decision problems**.

Problems that have a “yes” or “no” answer.

Why?

Theory reasons (ask me later).

But also most problems can be rephrased as very similar decision problems.

E.g. instead of “find the shortest path from s to t” ask
Is there a path from s to t of length at most k?

P

a set

Formally, question is whether algorithm exists, not whether it's known to humanity.

P (stands for "Polynomial")

The set of all decision problems that have an algorithm that runs in time $O(n^k)$ for some constant k.

The decision version of all problems we've solved in this class are in P.

P is an example of a "complexity class"

A set of problems that can be solved under some limitations (e.g. with some amount of memory or in some amount of time).

Remember the decision part! It's important

Be careful looking through old finals, prior quarters didn't include the decision requirement in the definition!

sorting an array $\in$ P

I'll know it when I see it.

Another class of problems we want to talk about.

"I'll know it when I see it" Problems.

Decision Problems such that:

If the answer is YES, you can prove the answer is yes by

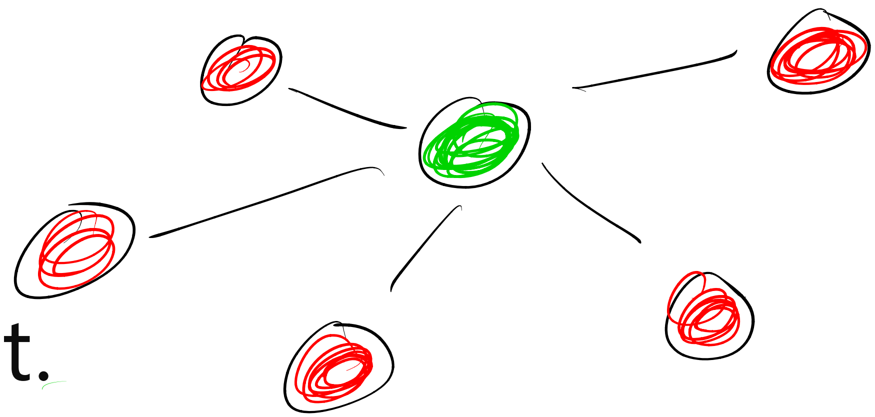
- Being given a "proof" or a "certificate"
- Verifying that certificate in polynomial time.

What certificate would be convenient for short paths?

- The path itself. Easy to check the path is really in the graph and really short.

certificate → you check quickly that it works

I'll know it when I see it.



More formally,

NP (stands for "nondeterministic polynomial")

The set of all decision problems such that if the answer is YES, there is a proof of that which can be verified in polynomial time.

Intuitively: you can show me why the answer is yes, and I can verify it.

3-COLOR is an NP problem

"Is there a Spanning Tree of cost at most 25?" is an NP problem

I'll know it when I see it.

More formally,

NP (stands for "nondeterministic polynomial")

The set of all decision problems such that if the answer is YES, there is a proof of that which can be verified in polynomial time.

Intuitively: you can show me why the answer is yes, and I can verify it.

3-COLOR is an NP problem

Give me the coloring (u is red, v is blue,…) and check each edge.

"Is there a spanning tree of cost at most 25?" is an NP problem

Give me the tree, I'll see if it's a spanning tree (run BFS, every vertex visited, no cycles, etc.); and see if the weight is small enough.

$P \overset{?}{=} NP$

I'll know it when I see it.

More formally,

NP (stands for "nondeterministic polynomial")

The set of all decision problems such that if the answer is YES, there is a proof of that which can be verified in polynomial time.

It's a common misconception that NP stands for "not polynomial"
Please never ever ever ever say that.

Please.

NP

We can **verify** YES instances of NP problems efficiently, but can we **decide** whether the answer is YES or NO efficiently?

That is, can we do it without the hint?

I.e. can you bootstrap the ability to check a certificate into the ability to find a certificate efficiently?

We don't know.

This is the P vs. NP problem.

P vs. NP

P (stands for "Polynomial")

The set of all decision problems that have an algorithm that runs in time $O(n^k)$ for some constant k.

NP (stands for "nondeterministic polynomial")

The set of all decision problems such that if the answer is YES, there is a proof of that which can be verified in polynomial time.

Claim: P $\subseteq$ NP (do you see why?)

$P \subseteq NP$ ✓

$NP \subseteq P$?

P vs. NP

Some problems in NP we know how to **solve** in polynomial time (solve from scratch, not just verify)

"Is there a spanning tree of cost at most 25?" can be solved with Prim's.

- It's in P.

P, NP

Other problems we don't know how to solve in polynomial time.

We don't know whether 3-COLOR is in P (most people don't think it is).

But maybe it is, and we just don't know the algorithm.

P vs. NP asks this question in general: does knowing you can verify a solution guarantee that you can find a solution?

will show up in EX 12

EXP

$\rightarrow O(2^{n^k})$

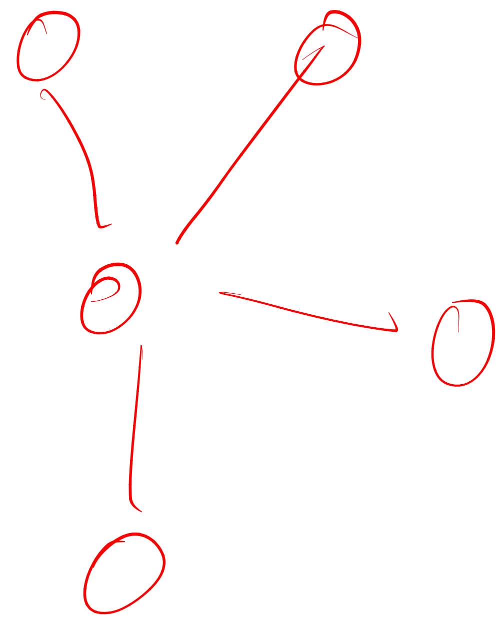
There is an algorithm to solve 3-COLOR, it's just slow

Think of a correct (just inefficient) algorithm to solve 3-COLOR

Generate all 3^n possible colorings, if one of them works, great! Return true.

If none of them work, return false.

This algorithm takes exponential time



EXP

EXP (stands for "Exponential")

The set of all decision problems that have an algorithm that runs in time $O\left(2^{n^{\wedge}k}\right)$ for some constant k.

3-COLOR is in EXP (we just saw why on the last slide)

So is

Claim: NP $\subseteq$ EXP (do you see why?)

$A \leq B$

Reductions

$NP \neq P$

Make sure you have the direction right, it's counter-intuitive!

Let's say we want to prove that some problem in NP needs exponential time (i.e. that P is not equal to NP).

Ideally we'd start with a really hard problem in NP.

What does it mean for one problem to be harder than another?

Polynomial Time Reducible

We say A reduces to B in polynomial time, if there is an algorithm that, using a (hypothetical) polynomial-time algorithm for B, solves problem A in polynomial-time.

I could solve problem A efficiently, if you give me a library that solves problem B efficiently

Reductions

Polynomial Time Reducible

We say A reduces to B in polynomial time, if there is an algorithm that, using a (hypothetical) polynomial-time algorithm for B, solves problem A in polynomial-time.

If A reduces to B then A should be “easier” than B. (for us as algorithm designers)

- If we can solve B, we can definitely solve A.

Usually denoted $A \leq_P B$.

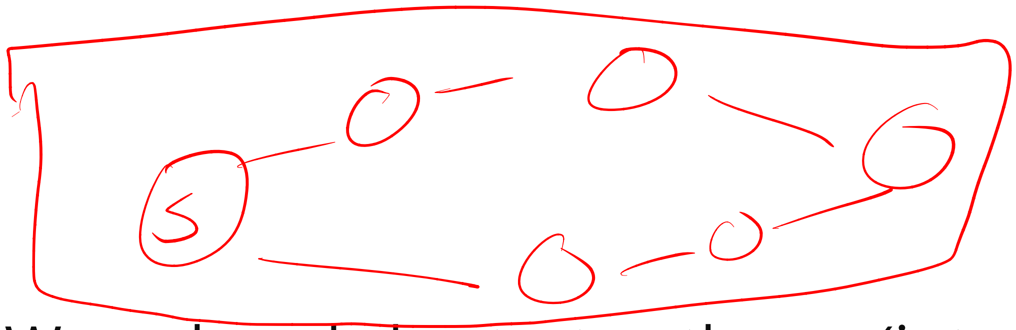
The Direction Matters!

Direction matters, and is often confusing:
$A \le B$ "A reduces to B"

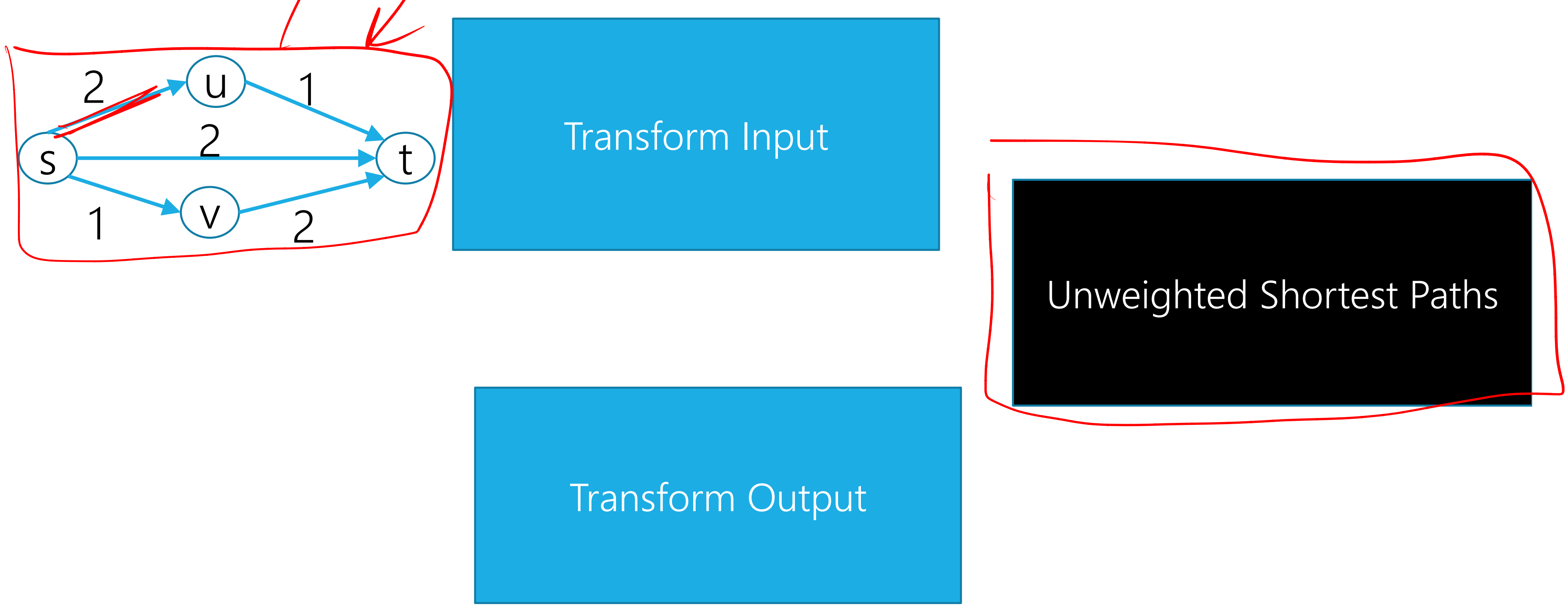
I wrote an algorithm to solve problem A using a library designed to solve problem B

"A is no harder than B" (solving B guarantees you can solve A, but maybe there's a different way to solve A)

How do you remember the direction?
repeat "Reduction from A to B means writing an algorithm for problem A using an algorithm designed for problem B" to yourself 50 times until it's stuck in your brain.



We reduced shortest paths on (integer-weighted) graphs to shortest paths on unweighted graphs



NP-complete

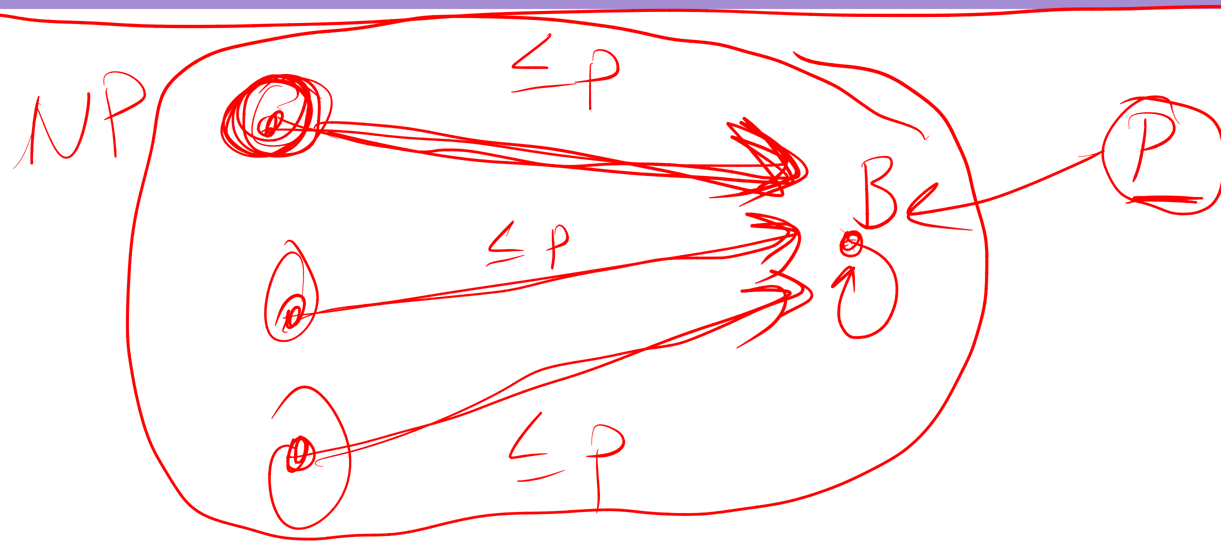
Let's say we want to prove that some problem in NP needs exponential time (i.e. that P is not equal to NP).

Ideally we'd start with a really hard problem in NP.

What is the hardest problem in NP?

NP-complete

A problem B is NP-complete if B is in NP and
for all problems A in NP, A reduces to B in polynomial time.



NP-complete

An NP-complete problem is a hardest problem in NP.

Seems like the right place to start for proving P≠NP.

It's also the right place to start for proving P=NP.

A polynomial time algorithm for one NP-complete problem, gives you a polynomial time algorithm for **every** problem in NP.

if an NP-complete problem is in P $\Rightarrow$ P = NP

oth $\Rightarrow$ P ≠ NP

Reductions Redux

Or think of it as a contrapositive:
If we have an algorithm for B, then we also have one for A.
If we don't (expect) to have an algorithm for A, then we don't (expect) to have one for B.

To show problem B is NP-hard

Reduce from A, a known NP-hard problem, to B.

From the known-hard problem to your new problem—must be that direction!

How do you remember the direction?
Robbie recommends you memorize "reduce from known problem to new problem" by repeating it to yourself 50 times.

Alternatively reconstruct that proof by contradiction from the last slide to see which direction is needed.

Examples

There are literally thousands of NP-complete problems.
And some of them look weirdly similar to problems we do know efficient algorithms for.

In P

Short Path

Given a directed graph, report if there is a path from s to t of length at most k.

$\leq$

NP-Complete

Long Path

Given a directed graph, report if there is a path from s to t of length at least k.

$\geq$

Examples

In P

Light Spanning Tree
Given a weighted graph, is there a spanning tree (a set of edges that connect all vertices) of weight at most k.

NP-Complete

Traveling Salesperson
Given a weighted graph, is there a tour (a walk that visits every vertex and returns to its start) of weight at most k.

The electric company just needs a greedy algorithm to lay its wires.
Amazon doesn't know a way to optimally route its delivery trucks.

Examples

In P

2-Coloring

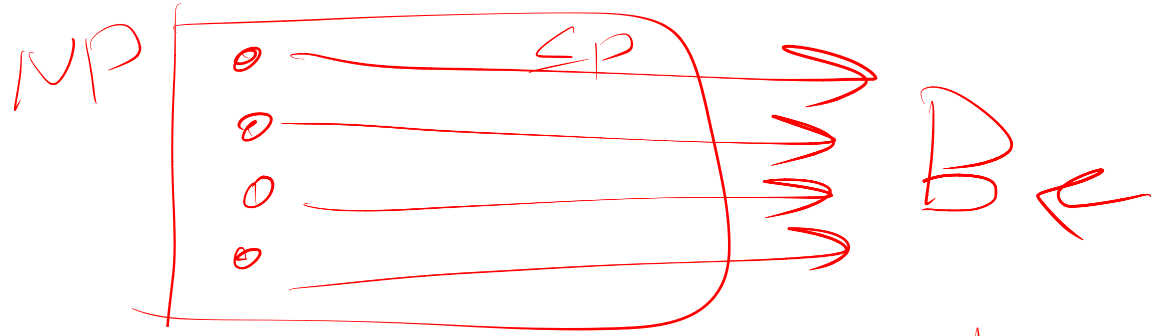
Given a graph, decide if you can color the vertices red and blue so every edge has different colored endpoints

NP-Complete

3-Coloring

Given a graph, decide if you can color the vertices red, blue, and green so every edge has different colored endpoints.

2-Coloring can be done with a modification of BFS (or DFS). Color the start vertex red, its neighbors must be blue, their neighbors red, etc.
No one knows how to tell if a graph is 3-colorable efficiently.



NP-hard

does not need to be ∈ NP itself

One more class:

NP-hard
Problem B is NP-hard if for all problems A in NP, A reduces to B in polynomial time.

An NP-hard problem need not be in NP.

Examples?

Find the "best possible" certificate for an NP-hard problem.

- Instead of a path of length at least k, find the longest path.
- Instead of a tour of weight at most k, find the shortest tour.

NP-hard

NP-hard

Problem B is NP-complete if
for all problems A in NP, A reduces to B in polynomial time.

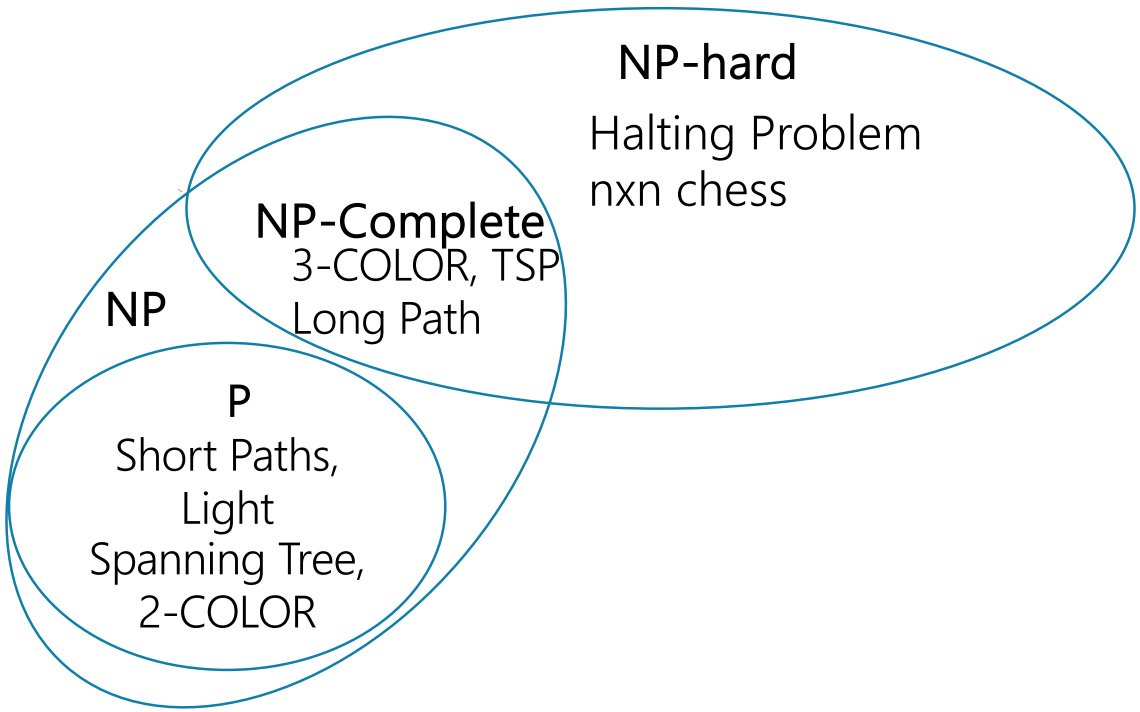
Other Examples: undecideable
The halting problem is NP-hard (but not NP-complete).
So is n x n chess.

n x n Chess

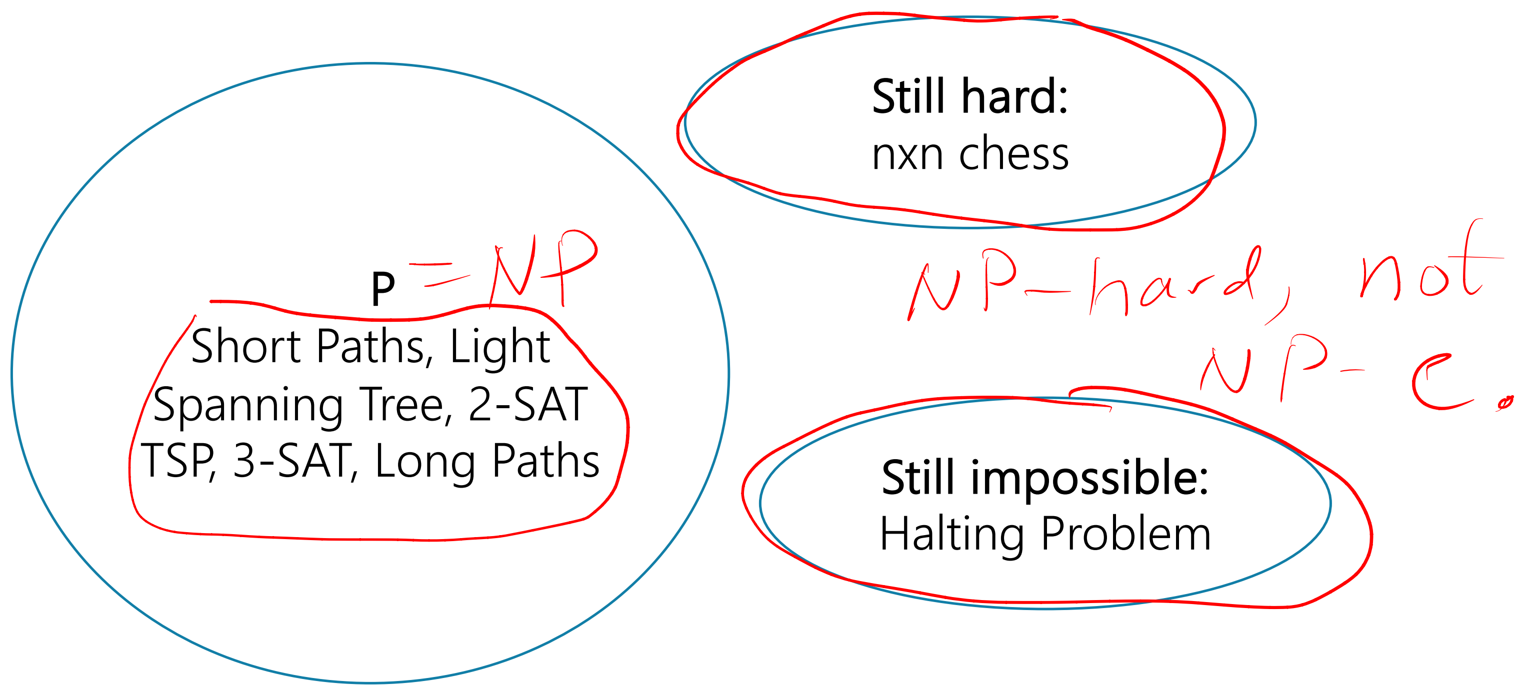
Given an n x n chessboard, can white force a win with perfect play?

$P \neq NP$

What The World Looks Like (We Think)



What The World Looks Like (If P=NP)



given a program and input to program → determine if it termates.

Why P vs. NP matters

P (stands for "Polynomial")

The set of all decision problems that have an algorithm that runs in time $O(n^k)$ for some constant k.

NP (stands for "nondeterministic polynomial")

The set of all decision problems such that if the answer is YES, there is a proof of that which can be verified in polynomial time.

NP-complete

Problem B is NP-complete if B is in NP and
for all problems A in NP, A reduces to B in polynomial time.

NP-hard

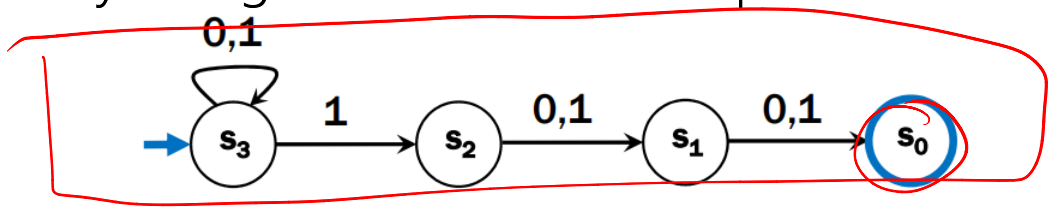
Problem B is NP-hard if
for all problems A in NP, A reduces to B in polynomial time.

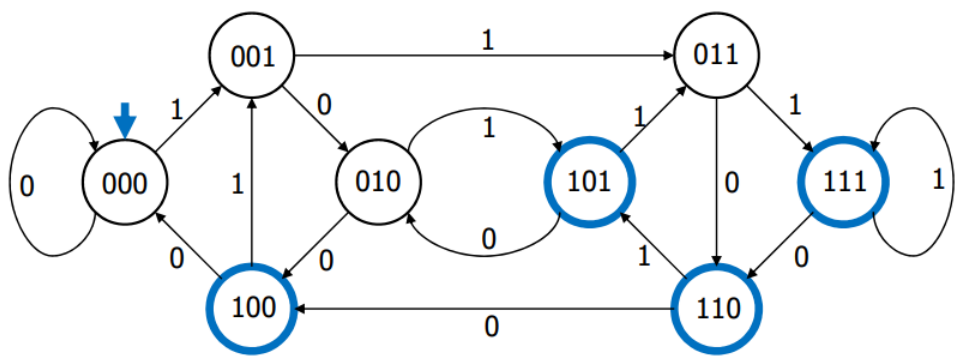
Why is it called NP?

You've seen nondeterministic computation before.
Back in 311.

NFAs would "magically" decide among a set of valid transitions.
Always choosing one that would lead to an accept state (if such a transition exists).

An NFA and a DFA for the language
"binary strings with a 1 in the 3rd position from the end."





From Kevin & Paul's 311 Lecture 23.

Nondeterminism

What would a nondeterministic **computer** look like?

It can run all the usual commands,

But it can also magically (i.e. nondeterministically) decide to set any bit of memory to 0 or 1.

Always choosing 0 or 1 to cause the computer to output YES,

(if such a choice exists).

If we had a nondeterministic computer…

Can you think of a polynomial time algorithm on a nondeterministic computer to:

Solve 2-COLOR?

Solve 3-COLOR?

If we had a nondeterministic computer…

Can you think of a polynomial time algorithm on a nondeterministic computer to:

Solve 2-COLOR?

Just run our regular deterministic polynomial time algorithm

Or nondeterministically guess colors, output if they work.

Solve 3-COLOR?

nondeterministically guess colors, output if they work.

Analogue of NFA/DFA equivalence

You showed in 311 that the set of languages decided by NFAs and DFAs were the same.

I.e. NFAs didn't let you solve more problems than DFAs.

But it did sometimes make the process a lot easier.

There are languages such that the best DFA is exponentially larger than the best NFA. (like the one from a few slides ago).

P vs. NP is an analogous question. Does non-determinism let us use exponentially fewer resources to solve some problems?

History, and Why P vs. NP?

NP-Completeness

If you find an efficient algorithm for an NP-complete problem, you have an algorithm for **every** problem in NP

Cook-Levin Theorem (1971)

SAT is NP-complete

Theorem 1: If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.

NP-Complete Problems

But Wait! There's more!

Main Theorem. All the problems on the following list are complete.

1. SATISFIABILITY
COMMENT: By duality, this problem is equivalent to determining whether a disjunctive normal form expression is a tautology.

2. 0-1 INTEGER PROGRAMMING
INPUT: integer matrix C and integer vector d
PROPERTY: There exists a 0-1 vector x such that $Cx = d$.

3. CLIQUE
INPUT: graph G, positive integer k
PROPERTY: G has a set of k mutually adjacent nodes.

4. SET PACKING
INPUT: Family of sets $\{S_j\}$, positive integer ℓ
PROPERTY: $\{S_j\}$ contains ℓ mutually disjoint sets.

5. NODE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: There is a set $R \subseteq N'$ such that $|R| \leq \ell$ and every arc is incident with some node in R.

6. SET COVERING
INPUT: finite family of finite sets $\{S_j\}$, positive integer k
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ containing $\leq k$ sets such that $\cup T_h = \cup S_j$.

7. FEEDBACK NODE SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $R \subseteq V$ such that every (directed) cycle of H contains a node in R.

8. FEEDBACK ARC SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $S \subseteq E$ such that every (directed) cycle of H contains an arc in S.

9. DIRECTED HAMILTON CIRCUIT
INPUT: digraph H
PROPERTY: H has a directed cycle which includes each node exactly once.

10. UNDIRECTED HAMILTON CIRCUIT
INPUT: graph G
PROPERTY: G has a cycle which includes each node exactly once.

11. SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
INPUT: Clauses $D_1, D_2, \ldots, D_r$, each consisting of at most 3 literals from the set $\{u_1, u_2, \ldots, u_m\} \cup \{\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_m\}$
PROPERTY: The set $\{D_1, D_2, \ldots, D_r\}$ is satisfiable.

12. CHROMATIC NUMBER
INPUT: graph G, positive integer k
PROPERTY: There is a function $\phi: N \to Z_k$ such that, if u and v are adjacent, then $\phi(u) \neq \phi(v)$.

13. CLIQUE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: N' is the union of ℓ or fewer cliques.

14. EXACT COVER
INPUT: family $\{S_j\}$ of subsets of a set $\{u_i, i = 1,2,\ldots,t\}$
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that the sets T_h are disjoint and $\cup T_h = \cup S_j = \{u_i, i = 1,2,\ldots,t\}$.

15. HITTING SET
INPUT: family $\{U_i\}$ of subsets of $\{s_j, j = 1,2,\ldots,r\}$
PROPERTY: There is a set W such that, for each i, $|W \cap U_i| = 1$.

16. STEINER TREE
INPUT: graph G, $R \subseteq N$, weighting function $w: A \to Z$, positive integer k
PROPERTY: G has a subtree of weight $\leq k$ containing the set of nodes in R.

17. 3-DIMENSIONAL MATCHING
INPUT: set $U \subseteq T \times T \times T$, where T is a finite set
PROPERTY: There is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate.

18. KNAPSACK
INPUT: $(a_1, a_2, \ldots, a_r, b) \in Z^{n+1}$
PROPERTY: $\Sigma a_j x_j = b$ has a 0-1 solution.

19. JOB SEQUENCING
INPUT: "execution time vector" $(T_1, \ldots, T_p) \in Z^p$,
"deadline vector" $(D_1, \ldots, D_p) \in Z^p$
"penalty vector" $(P_1, \ldots, P_p) \in Z^p$
positive integer k
PROPERTY: There is a permutation π of $\{1,2,\ldots,p\}$ such that

$$\left(\sum_{j=1}^{p} [\text{if } T_{\pi(1)} + \cdots + T_{\pi(j)} > D_{\pi(j)} \text{ then } P_{\pi(j)} \text{ else } 0]\right) \leq k \ .$$

20. PARTITION
INPUT: $(c_1, c_2, \ldots, c_s) \in Z^s$
PROPERTY: There is a set $I \subseteq \{1,2,\ldots,s\}$ such that

$$\sum_{h \in I} c_h = \sum_{h \notin I} c_h \ .$$

21. MAX CUT
INPUT: graph G, weighting function $w: A \to Z$, positive integer W
PROPERTY: There is a set $S \subseteq N$ such that

$$\sum_{\substack{\{u,v\} \in A \\ u \in S \\ v \notin S}} w(\{u,v\}) \geq W \ .$$

Karp's Theorem (1972)

A lot of problems people care about are NP-complete

NP-Complete Problems

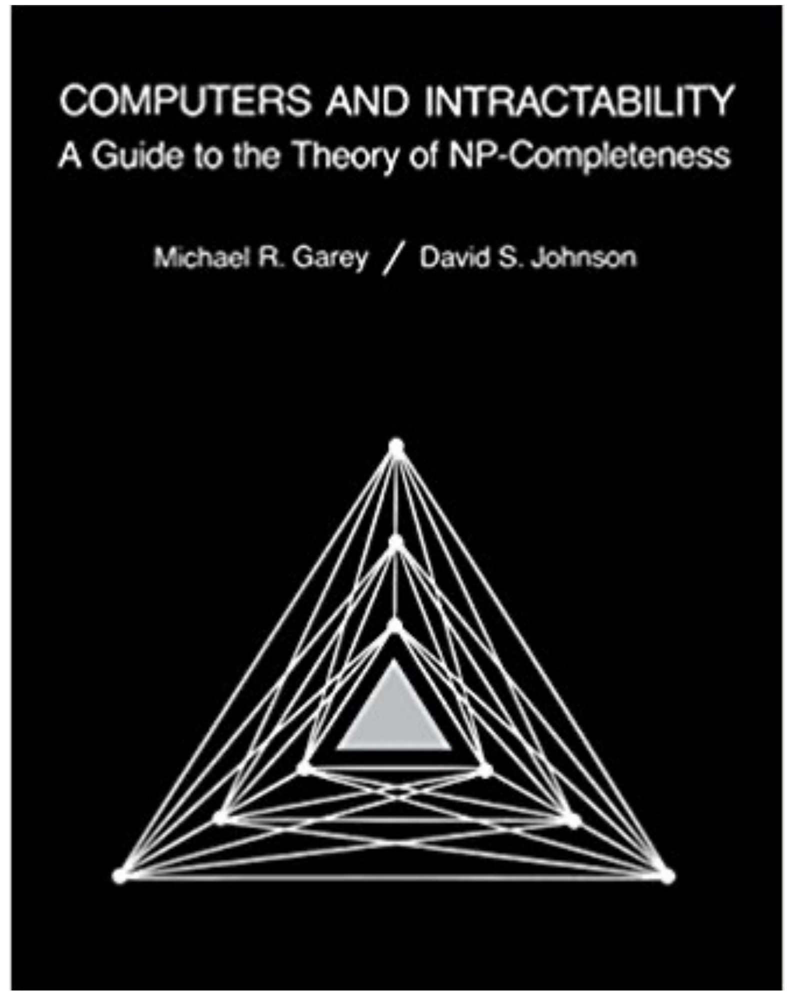
But Wait! There's more!

By 1979, at least 300 problems had been proven NP-complete.

Garey and Johnson put a list of all the NP-complete problems they could find in this textbook.

Took them almost 100 pages to just list them all.

No one has made a comprehensive list since.



NP-Complete Problems

But Wait! There's more!

In the last month, mathematicians and computer scientists have put papers on the arXiv claiming to show (at least) 10 more problems are NP-complete.

If you spend enough time trying to use computers to solve your problems, you will run into an NP-complete problem sooner or later.

What do you do?

Dealing with NP-Completeness

Option 1: Maybe it's a special case we understand

Maybe you don't need to solve the general problem, just a special case

- 2-COLOR vs. 3-COLOR

Option 2: Maybe it's a special case we *don't* understand (yet)

There are algorithms that are known to run quickly on "nice" instances. Maybe your problem has one of those.

One approach: Turn your problem into a SAT instance, find a solver and cross your fingers.

Dealing with NP-Completeness

Option 3: Approximation Algorithms

You might not be able to get an exact answer, but you might be able to get close.

Optimization version of Traveling Salesperson

Given a weighted graph, find a tour (a walk that visits every vertex and returns to its start) of minimum weight.

Algorithm:

Find a minimum spanning tree.

Have the tour follow the visitation order of a DFS of the spanning tree.

Theorem: This tour is at most twice as long as the best one.

Why should you care about P vs. NP

Most computer scientists are convinced that P≠NP.

Why should you care about this problem?

It's your chance for:

$1,000,000. The Clay Mathematics Institute will give $1,000,000 to whoever solves P vs. NP (or any of the 5 remaining problems they listed)

To get a Turing Award

Why should you care about P vs. NP

Most computer scientists are convinced that P≠NP.

Why should you care about this problem?

It's your chance for:

$1,000,000. The Clay Mathematics Institute will give $1,000,000 to whoever solves P vs. NP (or any of the 5 remaining problems they listed)

To get ~~a Turing Award~~ the Turing Award renamed after you.

Why Should You Care if P=NP?

Suppose P=NP.

Specifically that we found a genuinely in-practice efficient algorithm for an NP-complete problem. What would you do?

- $1,000,000 from the Clay Math Institute obviously, but what's next?

Why Should You Care if P=NP?

We found a genuinely in-practice efficient algorithm for an NP-complete problem. What would you do?

- Another $5,000,000 from the Clay Math Institute
- Put mathematicians out of work.
- Decrypt (essentially) all current internet communication.
- A world where P=NP is a very very different place from the world we live in now.

Why Should You Care if P≠NP?

We already expect P≠NP. Why should you care when we finally prove it?

P≠NP says something fundamental about the universe.

For some questions there is not a clever way to find the right answer

- Even though you'll know it when you see it.

Why Should You Care if P≠NP?

To prove P≠NP we need to better understand the differences between problems.

- Why do some problems allow easy solutions and others don't?
- What is the structure of these problems?

We don't care about P vs NP just because it has a huge effect about what the world looks like.

We will learn a lot about computation along the way.