Computational Complexity

- Classify problems according to the amount of computational resources used by the best algorithms that solve them
- Recall:
  - worst-case running time of an algorithm
  - max \# steps algorithm takes on any input of size \( n \)
- Define:
  - \( \text{TIME}(f(n)) \) to be the set of all decision problems solved by algorithms having worst-case running time \( O(f(n)) \)

Decision problems

- Computational complexity usually analyzed using decision problems
  - answer is just 1 or 0 (yes or no).
- Why?
  - much simpler to deal with
  - deciding whether \( G \) has a path from \( s \) to \( t \), is certainly no harder than finding a path from \( s \) to \( t \) in \( G \), so a lower bound on deciding is also a lower bound on finding
  - Less important, but if you have a good decider, you can often use it to get a good finder.

Polynomial time

- Define \( \mathcal{P} \) (polynomial-time) to be
  - the set of all decision problems solvable by algorithms whose worst-case running time is bounded by some polynomial in the input size.
  \[ \mathcal{P} = \bigcup_{k \geq 0} \text{TIME}(n^k) \]

Beyond \( \mathcal{P} \)?

- There are many natural, practical problems for which we don’t know any polynomial-time algorithms
- e.g. decisionTSP:
  - Given a weighted graph \( G \) and an integer \( k \), does there exist a tour that visits all vertices in \( G \) having total weight at most \( k \)?

Relative Complexity of Problems

- Want a notion that allows us to compare the complexity of problems
- Want to be able to make statements of the form
  - “If we could solve problem \( R \) in polynomial time then we can solve problem \( L \) in polynomial time”
  - “Problem \( R \) is at least as hard as problem \( L \)”
Polynomial Time Reduction

- $L \leq_p R$ if there is an algorithm for $L$ using a ‘black box’ (subroutine) that solves $R$ that
  - Uses only a polynomial number of steps
  - Makes only a polynomial number of calls to a subroutine for $R$

Thus, poly time algorithm for $R$ implies poly time algorithm for $L$

If you can prove there is no fast algorithm for $L$, then that proves there is no fast algorithm for $R$.

A Special kind of Polynomial-Time Reduction

- We will always use a restricted form of polynomial-time reduction often called Karp or many-one reduction.

- $L \leq_M^1 R$ if and only if there is an algorithm for $L$ given a black box solving $R$ that on input $x$
  - Runs for polynomial time computing an input $T(x)$
  - Makes one call to the black box for $R$
  - Returns the answer that the black box gave

We say that the function $T$ is the reduction.

Why the name reduction?

- Weird: it maps an easier problem into a harder one.

- Solving partial differential equations in general is a much harder problem than solving E&M problems.

A geek joke

An engineer
- is placed in a kitchen with an empty kettle on the table and told to boil water; she fills the kettle with water, puts it on the stove, turns on the gas and boils water.
- she is next confronted with a kettle full of water sitting on the counter and told to boil water; she puts it on the stove, turns on the gas and boils water.

A mathematician
- is placed in a kitchen with an empty kettle on the table and told to boil water; he fills the kettle with water, puts it on the stove, turns on the gas and boils water.
- he is next confronted with a kettle full of water sitting on the counter and told to boil water: he empties the kettle in the sink, places the empty kettle on the table and says, ‘I’ve reduced this to an already solved problem’.

Reductions from a Special Case to a General Case

Show: Vertex-Cover $\leq_p$ Set-Cover

- Vertex-Cover:
  - Given an undirected graph $G=(V,E)$ and an integer $k$ is there a subset $W$ of $V$ of size at most $k$ such that every edge of $G$ has at least one endpoint in $W$? (i.e. $W$ covers all edges of $G$).

- Set-Cover:
  - Given a set $U$ of $n$ elements, a collection $S_1,...,S_m$ of subsets of $U$, and an integer $k$, does there exist a collection of at most $k$ sets whose union is equal to $U$?

The Simple Reduction

- Transformation $T$ maps $(G=(V,E),k)$ to $(U,S_1,...,S_m,k')$
  - $U\leftarrow E$
  - For each vertex $v \in V$ create a set $S_v$ containing all edges that touch $v$
  - $k'\leftarrow k$

- Reduction $T$ is clearly polynomial-time to compute

- We need to prove that the resulting algorithm gives the right answer!
Proof of Correctness

Two directions:
- If the answer to Vertex-Cover on \((G,k)\) is YES then the answer for Set-Cover on \(T(G,k)\) is YES
  - If a set \(W\) of \(k\) vertices covers all edges then the collection \(\{S_v \mid v \in W\}\) of \(k\) sets covers all of \(U\)
- If the answer to Set-Cover on \(T(G,k)\) is YES then the answer for Vertex-Cover on \((G,k)\) is YES
  - If a subcollection \(S_{v_1}, \ldots, S_{v_k}\) covers all of \(U\) then the set \(\{v_1, \ldots, v_k\}\) is a vertex cover in \(G\).

Reductions by Simple Equivalence

Show: Independent-Set \(\leq_p\) Clique

Independent-Set:
- Given a graph \(G=(V,E)\) and an integer \(k\), is there a subset \(U\) of \(V\) with \(|U| \geq k\) such that no two vertices in \(U\) are joined by an edge.

Clique:
- Given a graph \(G=(V,E)\) and an integer \(k\), is there a subset \(U\) of \(V\) with \(|U| \geq k\) such that every pair of vertices in \(U\) is joined by an edge.

Independent-Set \(\leq_p\) Clique

Show: Independent Set \(\leq_p\) Vertex-Cover

Vertex-Cover:
- Given an undirected graph \(G=(V,E)\) and an integer \(k\), is there a subset \(W\) of \(V\) of size at most \(k\) such that every edge of \(G\) has at least one endpoint in \(W\)? (i.e. \(W\) covers all edges of \(G\)).

Independent-Set:
- Given a graph \(G=(V,E)\) and an integer \(k\), is there a subset \(U\) of \(V\) with \(|U| \geq k\) such that no two vertices in \(U\) are joined by an edge.

Reduction Idea

Claim: In a graph \(G=(V,E)\), \(S\) is an independent set iff \(V-S\) is a vertex cover

Proof:
- \(\Rightarrow\) Let \(S\) be an independent set in \(G\)
  - Then \(S\) contains at most one endpoint of each edge of \(G\)
  - At least one endpoint must be in \(V-S\)
  - \(V-S\) is a vertex cover
- \(\Leftarrow\) Let \(W=V-S\) be a vertex cover of \(G\)
  - Then \(S\) does not contain both endpoints of any edge (else \(W\) would miss that edge)
  - \(S\) is an independent set

Reduction

- Map \((G,k)\) to \((G,n-k)\)
  - Previous lemma proves correctness
  - Clearly polynomial time
  - We also get that \(\text{Vertex-Cover} \leq_p \text{Independent Set}\)
Satisfiability

- Boolean variables $x_1,...,x_n$
  - taking values in $\{0,1\}$. $0=\text{false}$, $1=\text{true}$
- Literals
  - $x_i$ or $\neg x_i$ for $i=1,...,n$
- Clause
  - a logical OR of one or more literals
  - e.g. $(x_1 \lor \neg x_3 \lor x_7 \lor x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses

Satisfiability

- CNF formula example
  - $(x_1 \lor \neg x_3 \lor x_7 \lor x_{12}) \land (x_2 \lor \neg x_4 \lor x_7 \lor x_3)$
- If there is some assignment of 0’s and 1’s to the variables that makes it true then we say the formula is satisfiable

Common property of these problems

- There is a special piece of information, a short certificate or proof, that allows you to efficiently verify (in polynomial-time) that the YES answer is correct. This certificate might be very hard to find

- e.g. DecisionTSP: the tour itself, Independent-Set, Clique: the set $U$ Satisfiability: an assignment that makes $F$ true.

The complexity class NP

NP consists of all decision problems where

- You can verify the YES answers efficiently (in polynomial time) given a short (polynomial-size) certificate

And

- No certificate can fool your polynomial time verifier into saying YES for a NO instance

More Precise Definition of NP

- A decision problem is in NP iff there is a polynomial time procedure verify($\ldots$), and an integer $k$ such that
  - for every input $x$ to the problem that is a YES instance there is a certificate $t$ with $|t| \leq |x|^k$ such that verify($x,t$) = YES and
  - for every input $x$ to the problem that is a NO instance there does not exist a certificate $t$ with $|t| \leq |x|^k$ such that verify($x,t$) = YES

Example: CLIQUE is in NP

procedure verify($x,t$)
  if
    $x$ is a well-formed representation of a graph $G=(V,E)$ and an integer $k$,
    and
    $t$ is a well-formed representation of a vertex subset $U$ of $V$ of size $k$,
    and
    $U$ is a clique in $G$,
    then output "YES"
  else output "I'm unconvinced"
Is it correct?

For every \( x = (G,k) \) such that \( G \) contains a \( k \)-clique, there is a certificate \( t \) that will cause \( \text{verify}(x,t) \) to say \( \text{YES} \).

- \( t \) = a list of the vertices in such a \( k \)-clique

And no certificate can fool \( \text{verify}(x,\cdot) \) into saying \( \text{YES} \) if either

- \( x \) isn't well-formed (the uninteresting case)
- \( x = (G,k) \) but \( G \) does not have any cliques of size \( k \) (the interesting case)

Keys to showing that a problem is in NP

- What's the output? (must be YES/NO)
- What must the input look like?
- Which inputs need a \( \text{YES} \) answer?
  - Call such inputs \( \text{YES} \) inputs/\( \text{YES} \) instances
- For every given \( \text{YES} \) input, is there a certificate that would help?
  - \( \text{OK if some inputs need no certificate} \)
- For any given \( \text{NO} \) input, is there a fake certificate that would trick you?

Solving NP problems without hints

- The only obvious algorithm for most of these problems is brute force:
  - try all possible certificates and check each one to see if it works.
  - Exponential time:
    - \( 2^n \) truth assignments for \( n \) variables
    - \( n! \) possible TSP tours of \( n \) vertices
    - \( \binom{n}{k} \) possible \( k \) element subsets of \( n \) vertices
    - etc.

What We Know

- Nobody knows if all problems in \( \text{NP} \) can be done in polynomial time, i.e. does \( \text{P} = \text{NP} \)?
  - one of the most important open questions in all of science.
  - huge practical implications
- Every problem in \( \text{P} \) is in \( \text{NP} \)
  - one doesn't even need a certificate for problems in \( \text{P} \) so just ignore any hint you are given
- Every problem in \( \text{NP} \) is in exponential time

P and NP

NP-hardness & NP-completeness

- Some problems in \( \text{NP} \) seem hard
  - people have looked for efficient algorithms for them for hundreds of years without success
- However
  - nobody knows how to prove that they are really hard to solve, i.e. \( \text{P} \neq \text{NP} \)
Problems in NP that seem hard

- Some Examples in NP
  - Satisfiability
  - Independent-Set
  - Clique
  - Vertex Cover
- All hard to solve; certificates seem to help on all
- Fast solution to any gives fast solution to all!

NP-hardness & NP-completeness

- Alternative approach to proving problems not in P
  - show that they are at least as hard as any problem in NP
- Rough definition:
  - A problem is NP-hard iff it is at least as hard as any problem in NP
  - A problem is NP-complete iff it is both
    - NP-hard
    - in NP

P and NP

- NP-hard
- NP-complete

Cook's Theorem

- Theorem (Cook 1971): Satisfiability is NP-complete
- Recall
  - CNF formula
    - e.g. \((x_1 \lor \neg x_3 \lor x_7 \lor x_{12}) \land (x_2 \lor \neg x_4 \lor x_8 \lor x_9)\)
  - If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable
- Satisfiability: Given a CNF formula \(F\), is it satisfiable?

Implications of Cook's Theorem?

- There is at least one interesting super-hard problem in NP
- Is that such a big deal?
  - YES!
    - There are lots of other problems that can be solved if we had a polynomial-time algorithm for Satisfiability
    - Many of these problems are exactly as hard as Satisfiability
A useful property of polynomial-time reductions

Theorem: If \( L \leq_{p} R \) and \( R \leq_{p} S \) then \( L \leq_{p} S \)

- Proof idea: (Using \( <_{p} <_{p} \) )
  - Compose the reduction \( T \) from \( L \) to \( R \) with the reduction \( T' \) from \( R \) to \( S \) to get a new reduction \( T''(x) = T(T'(x)) \) from \( L \) to \( S \).

The general case is similar and uses the fact that the composition of two polynomials is also a polynomial.

Cook’s Theorem & Implications

- Theorem (Cook 1971): Satisfiability is NP-complete
  - For proof see CSE 431
- Corollary: \( R \) is NP-hard \( \iff \) Satisfiability \( \leq_{p} R \)
  - (or \( Q \leq_{p} R \) for any NP-complete problem \( Q \))

Proof:
- If \( R \) is NP-hard then every problem in NP polynomial-time reduces to \( R \), in particular Satisfiability does since it is in NP.
- For any problem \( L \) in NP, \( L \leq_{p} \) Satisfiability and so if Satisfiability \( \leq_{p} R \) we have \( L \leq_{p} R \).
- Therefore \( R \) is NP-hard if Satisfiability \( \leq_{p} R \).

Another NP-complete problem: Satisfiability \( \leq_{p} \) Independent-Set

A Tricky Reduction:
- mapping CNF formula \( F \) to a pair \(<G,k>\)
- Let \( m \) be the number of clauses of \( F \)
- Create a vertex in \( G \) for each literal in \( F \)
- Join two vertices \( u, v \) in \( G \) by an edge iff \( u \) and \( v \) correspond to literals in the same clause of \( F \), (green edges) or \( u \) and \( v \) correspond to literals \( x \) and \( \neg x \) (or vice versa) for some variable \( x \). (red edges).
- Set \( k = m \)
- Clearly polynomial-time

Satisfiability \( \leq_{p} \) Independent-Set

Correctness:
- If \( F \) is satisfiable then there is some assignment that satisfies at least one literal in each clause.
- Consider the set \( U \) in \( G \) corresponding to the first satisfied literal in each clause.
  - \( |U| = m \)
  - Since \( U \) has only one vertex per clause, no two vertices in \( U \) are joined by green edges.
  - Since a truth assignment never satisfies both \( x \) and \( \neg x \), \( U \) doesn’t contain vertices labeled both \( x \) and \( \neg x \) and so no vertices in \( U \) are joined by red edges.
  - Therefore \( G \) has an independent set, \( U \), of size at least \( m \).
- Therefore \( (G,m) \) is a YES for independent set.

Satisfiability \( \leq_{p} \) Independent-Set

Given assignment \( x_1 = x_2 = x_3 = x_4 = 1 \), \( U \) is as circled
Satisfiability \(\leq^p\) Independent-Set

- Correctness continued:
  - If \((G, m)\) is a YES for Independent-Set then there is a set \(U\) of \(m\) vertices in \(G\) containing no edge.
  - Therefore \(U\) has precisely one vertex per clause because of the green edges in \(G\).
  - Because of the red edges in \(G\), \(U\) does not contain vertices labeled both \(x\) and \(\overline{x}\).

- By construction, \(A\) satisfies \(F\).
- Therefore \(F\) is a YES for Satisfiability.

Independent-Set is NP-complete

- We just showed that Independent-Set is NP-hard and we already knew Independent-Set is in NP.
- Corollary: Clique is NP-complete
  - We showed already that Independent-Set \(\leq_p\) Clique and Clique is in NP.

Is NP as bad as it gets?

- NO! NP-complete problems are frequently encountered, but there's worse:
  - Some problems provably require exponential time.
  - Ex: Does \(P\) halt on \(x\) in \(2^{n^2}\) steps?
  - Some require \(2^n\), \(2^{n^2}\), \(2^{2^n}\), ... steps
  - And of course, some are just plain uncomputable

Problems we already know are NP-complete

- Satisfiability
- Independent-Set
- Clique
- Vertex-Cover

- There are 1000's of practical problems that are NP-complete, e.g. scheduling, optimal VLSI layout etc.

Steps to Proving Problem R is NP-complete

- Show R is NP-hard:
  - State: Reduction is from NP-hard Problem L
  - Show what the map \(T\) is
  - Argue that \(T\) is polynomial time
  - Argue correctness: two directions Yes for L implies Yes for R and vice versa.
- Show R is in NP
  - State what hint is and why it works
  - Argue that it is polynomial-time to check.
A particularly useful problem for proving NP-completeness

- 3-SAT: Given a CNF formula $F$ having precisely 3 variables per clause (i.e., in 3-CNF), is $F$ satisfiable?

- Claim: 3-SAT is NP-complete

- Proof:
  - 3-SAT is NP
    - Certificate is a satisfying assignment
    - Just like Satisfiability it is polynomial-time to check the certificate

Satisfiability $\leq_p$ 3-SAT

- Goal:
  - An assignment $a$ to the original variables makes clause $C$ true in $F$ if:
    - there is an assignment to the extra variables that together with the assignment $a$ will make all new clauses corresponding to $C$ true.
  - Define the reduction clause-by-clause
  - We'll use variable names $z_i$ to denote the extra variables related to a single clause $C$ to simplify notation
    - In reality, two different original clauses will not share $z_i$

- Reduction:
  - Map CNF formula $F$ to another CNF formula $G$ that has precisely 3 variables per clause.
    - $G$ has one or more clauses for each clause of $F$
    - $G$ will have extra variables that don't appear in $F$
      - For each clause $C$ of $F$ there will be a different set of variables that are used only in the clauses of $G$ that correspond to $C$
  - Use two new variables $(z_1, z_2)$ and put two clauses in $G$ that correspond to $C$
  - If original $C$ is true under assignment $a$ then both new clauses will be true under $a$
  - If new clauses are both true under some assignment $b$ then the value of $a$ doesn't help in one of the two clauses so $C$ must be true under $b$

Satisfiability $\leq_p$ 3-SAT

- If $C$ has 1 variable: e.g. $C=x_i$
  - Use two new variables $z_1, z_2$ and put 4 new clauses in $G$
    - $(x_i \lor \neg z_1 \lor \neg z_2) \land (x_i \lor \neg z_1 \lor z_2) \land (x_i \lor z_1 \lor \neg z_2) \land (x_i \lor z_1 \lor z_2)$
  - If original $C$ is true under assignment $a$ then all new clauses will be true under $a$
  - If new clauses are all true under some assignment $b$ then the values of $z_1$ and $z_2$ don’t help in one of the 4 clauses so $C$ must be true under $b$

Satisfiability $\leq_p$ 3-SAT

- If $C$ has $k \geq 4$ variables: e.g. $C=(x_1 \lor \ldots \lor x_k)$
  - Use $k-3$ new variables $z_1, \ldots, z_{k-2}$ and put $k-2$ new clauses in $G$
    - $(x_i \lor x_j \lor z_1) \land (\neg z_1 \lor x_j \lor z_2) \land (\neg z_2 \lor x_j \lor z_3) \land \ldots \land (\neg z_{k-3} \lor x_j \lor z_{k-2}) \land (\neg z_{k-2} \lor x_j \lor x_{k-1})$
  - If original $C$ is true under assignment $a$ then some $x_i$ is true for $1 \leq i \leq k$. By setting $z_j$ true for all $j > i$, we can extend $a$ to make all new clauses true.
  - If new clauses are all true under some assignment $b$ then some $x_i$ must be true for $1 \leq i \leq k$ because $z_j \lor (\neg z_j \lor z_{j+1}) \land \ldots \land (\neg z_{k-2} \lor z_{k-1}) \land \neg z_{k-1}$ is not satisfiable
**Graph Colorability**

- **Defn:** Given a graph $G=(V,E)$, and an integer $k$, a **$k$-coloring** of $G$ is an assignment of up to $k$ different colors to the vertices of $G$ so that the endpoints of each edge have different colors.

- **3-Color:** Given a graph $G=(V,E)$, does $G$ have a 3-coloring?

- **Claim:** 3-Color is NP-complete

- **Proof:** 3-Color is in NP:
  - Hint is an assignment of red, green, blue to the vertices of $G$
  - Easy to check that each edge is colored correctly

**3-SAT $\leq P 3$-Color**

- **Reduction:**
  - We want to map a 3-CNF formula $F$ to a graph $G$ so that $G$ is 3-colorable iff $F$ is satisfiable

- **Base Triangle**

- **Variable Part:**
  - in 3-coloring, variable colors correspond to some truth assignment (same color as $T$ or $F$)

- **Clause Part:**
  - Add one 6 vertex gadget per clause connecting its ‘outer vertices’ to the literals in the clause

- **Any truth assignment satisfying the formula can be extended to a 3-coloring of the graph**
3-SAT $\leq_p$ 3-Color

Any 3-coloring of the graph colors each gadget triangle using each color

3-SAT $\leq_p$ 3-Color

Any 3-coloring of the graph has an F opposite the O color in the triangle of each gadget

3-SAT $\leq_p$ 3-Color

Any 3-coloring of the graph has T at the other end of the blue edge connected to the F

3-SAT $\leq_p$ 3-Color

Any 3-coloring of the graph yields a satisfying assignment to the formula

More NP-completeness

- Subset-Sum problem
  - Given $n$ integers $w_1, ..., w_n$ and integer $W$
  - Is there a subset of the $n$ input integers that adds up to exactly $W$?
  - $O(nW)$ solution from dynamic programming but if $W$ and each $w_i$ can be $n$ bits long then this is exponential time

- 3-SAT $\leq_p$ Subset-Sum
  - Given a 3-CNF formula with $m$ clauses and $n$ variables
  - Will create $2m+2n$ numbers that are $m+n$ digits long
    - Two numbers for each variable $x_i$
      - $t_i$ and $f_i$ (corresponding to $x_i$ being true or false)
    - Two extra numbers for each clause
      - $u_j$ and $v_j$ (filler variables to handle number of false literals in clause $C_j$)
3-SAT $\leq_p$ Subset-Sum

\[
C_{\phi}(x_1, \ldots, x_n, x_j)
\]

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
</tr>
<tr>
<td>t_2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
</tr>
<tr>
<td>t_3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>1</td>
</tr>
<tr>
<td>t_4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>\ldots</td>
<td>0</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>u_jv_i &amp; 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>u_jv_k &amp; 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>W &amp; 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>\ldots</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>\ldots</td>
<td>3</td>
</tr>
</tbody>
</table>

P vs NP

**Theory**
- **P = NP?**
- Open Problem!
- Bet against it

**Practice**
- Many interesting, useful, natural, well-studied problems known to be NP-complete
- With rare exceptions, no one routinely succeeds in finding exact solutions to large, arbitrary instances