The Romans didn't find algebra very challenging, because X was always 10.

NP-Completeness

Brute-Force Solution: $O(n!)$

Dynamic Programming Algorithms: $O(n^2 2^n)$

Selling on eBay: $O(1)$

Still working on your route?

Shut the hell up.
Group of problems whose complexity is related to all others in the group.

Phenomenon is called **NP-Complete**.

If a polynomial time algorithm could solve *one* of these problems, then it could solve *all* other problems in the group.

Examples:

The Satisfiability Problem

Consider the Boolean operations **AND**, **OR**, and **NOT**, as shown:

<table>
<thead>
<tr>
<th>$0 \land 0 = 0$</th>
<th>$0 \lor 0 = 0$</th>
<th>$\overline{0} = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \land 1 = 0$</td>
<td>$0 \lor 1 = 1$</td>
<td>$\overline{1} = 0$</td>
</tr>
<tr>
<td>$1 \land 0 = 0$</td>
<td>$1 \lor 0 = 1$</td>
<td></td>
</tr>
<tr>
<td>$1 \land 1 = 1$</td>
<td>$1 \lor 1 = 1$</td>
<td></td>
</tr>
</tbody>
</table>

Consider the Boolean formula:

$$\phi = (\overline{x} \land y) \lor (x \land \overline{z})$$

The formula is **Satisfiable** if some assignment of 0s and 1s to the formula makes $\phi = 1$.

The **Satisfiability Problem** tests whether a Boolean formula is satisfiable.

$\text{SAT} = \{ \langle \varphi \rangle \mid \varphi \text{ is a satisfiable Boolean formula} \}$

**THEOREM:**

“$\text{SAT} \in \text{P iff } \text{P} = \text{NP}$.”

The logic behind this requires a lot more work.
**Polynomial Time Reducibility**

**DEFINITION:**
A function $f: \Sigma^* \rightarrow \Sigma^*$ is a **polynomial time computable function** if some polynomial time Turing machine $M$ exists that halts with just $f(w)$ on its tape, when started on any input $w$.

**DEFINITION:**
Language $A$ is **polynomial time mapping reducible**, or simply polynomial time reducible, to language $B$, written $A \leq_p B$, if a polynomial time computable function $f: \Sigma^* \rightarrow \Sigma^*$ exists, where for every $w$, $w \in A \iff f(w) \in B$. The function $f$ is called the **polynomial time reduction** of $A$ to $B$.

“polynomial time mapping reducible” is also known as “polynomial time many–one reducibility”

**Intuition:** using one problem to solve another.
(sound familiar?)

Sad Reduction Face
Theorem

“If $A \leq_p B$ and $B \in P$, then $A \in P$.”

PROOF:
Let $M$ be the polynomial time algorithm deciding $B$ and $f$ be the polynomial time reduction from $A$ to $B$. We describe a polynomial time algorithm $N$ deciding $A$ as follows.

$N = “$On input $w:$

1. Compute $f(w)$.
2. Run $M$ on input $f(w)$ and output whatever $M$ outputs.”

We have $w \in A$ whenever $f(w) \in B$ because $f$ is a reduction from $A$ to $B$.

Thus, $M$ accepts $f(w)$ whenever $w \in A$.

Moreover, $N$ runs in polynomial time because each of its two stages runs in polynomial time. Note that stage 2 runs in polynomial time because the composition of two polynomials is a polynomial.

Proof complete.
3SAT

3SAT is SAT in which all formulas are in a specific format (analogous to Chomsky Normal Form for CFGs).

- A literal is a Boolean variable ($x$ or $\neg x$).
- A clause is several literals connected with $\lor$s.
- A Boolean formula in conjunctive normal form (called a cnf-formula) connects multiple clauses with $\land$s.
- A 3cnf-formula is when all clauses contain exactly 3 literals. Example:

$$(x_1 \lor \overline{x}_2 \lor x_3) \land (x_3 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor \overline{x}_6 \lor x_4) \land (x_4 \lor x_5 \lor \overline{x}_6)$$

Let 3SAT = \{ $\langle \phi \rangle$ | $\phi$ is a satisfiable 3cnf-formula $\}$.

Can you convert the following to 3cnf? Remember De Morgan’s laws.

- $a \land (b \lor c \lor d)$
  - $(a \lor a \lor a) \land (b \lor c \lor d)$
- $\neg(b \lor c)$
  - $\neg b \land \neg c$
  - $(\neg b \lor \neg b \lor \neg b) \land (\neg c \lor \neg c \lor \neg c)$
- $\neg(b \lor (a \land c))$
  - $\neg b \land \neg(a \land c)$
  - $\neg b \land (\neg a \lor \neg c)$
  - $(\neg b \lor \neg b \lor \neg b) \land (\neg a \lor \neg c \lor \neg c)$
- $\neg(b \land (a \lor c))$
  - $\neg b \land \neg(a \lor c)$
  - $\neg b \land (\neg a \land \neg c)$
  - $(\neg b \lor \neg a \lor \neg a) \land (\neg b \lor \neg c \lor \neg c)$

See Tseytin Transformations.
Theorem: 3SAT Reduces to CLIQUE

“3SAT is polynomial time reducible to CLIQUE.”

IDEA:
Give a reduction to convert formulas to graphs.

Let $\phi$ be a formula with $k$ clauses such as

$\phi = (a_1 \lor b_1 \lor c_1) \land (a_2 \lor b_2 \lor c_2) \land \ldots \land (a_k \lor b_k \lor c_k)$.

Reduction $f$ generates the string $(G, k)$ where $G$ is an undirected graph.

Nodes of $G$ are arranged in $k$ groups of 3 (each called a **triple**). Each node corresponds to a literal in a single triple.

There is an edge between all node pairs, except those with opposite labels ($x$ and $\neg x$), and nodes within the same triple.
Theorem: 3SAT Reduces to CLIQUE (cont.)

PROOF:

“φ is satisfiable iff G has a k-clique.”

Suppose φ has a satisfying argument. At least one literal must be true in every clause. Selecting one true literal from each clause will form a k-clique in the graph. k nodes were selected because we only chose one from each triple. Each pair is joined with an edge because it does not meet the exception given earlier. Therefore, G contains a k-clique.

Conversely, suppose G contains a k-clique. Each node must belong to a different triple. A node’s label must be true. Contradictory labels cannot be connected. Therefore, the k-clique contains a node from each clause, and each node’s label is true. All clauses from φ are true, and thus φ is satisfied.

\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2) \]
DEFINITION:
A language $B$ is NP-complete if it satisfies two conditions:
1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

THEOREM (e.g., Consequences):
“If $B$ is NP-complete and $B \in P$, then $P = NP$.”

Proof follows directly from definition of polynomial time reducibility.

Intuition: If CLIQUE is solvable in polynomial time, so is 3SAT!
Theorem

“If \( B \) is NP-complete and \( B \leq_p C \) for \( C \) in NP, then \( C \) is NP-complete.”

PROOF:

\( C \) is in NP. We show that every \( A \) in NP is polynomial time reducible to \( C \).

Because \( B \) is NP-complete, every language in NP is polynomial time reducible to \( B \), and \( B \) in turn is polynomial time reducible to \( C \).

Polynomial time reductions compose; that is, if \( A \) is polynomial time reducible to \( B \) and \( B \) is polynomial time reducible to \( C \), then \( A \) is polynomial time reducible to \( C \).

Hence every language in NP is polynomial time reducible to \( C \).
THEOREM: “SAT is NP-Complete.”

Part 1: SAT is in NP.

Easy. A nondeterministic polynomial time machine can guess an assignment to a given formula φ and accept if the assignment satisfies φ.

Part 2: Every A in NP is polynomial time reducible to SAT.
Cook-Levin Theorem (cont.)

Let $N$ be a nondeterministic Turing machine that decides $A$ in $n^k$ time for some constant $k$.

A tableau for $N$ on $w$ is an $n^k \times n^k$ table whose rows are the configurations of a branch of the computation of $N$ on input $w$.

For convenience, first and last columns are $\#$s.

A tableau is accepting if any row of the tableau is an accepting configuration.

Every accepting tableau for $N$ on $w$ corresponds to an accepting computation branch of $N$ on $w$.

Thus, the problem of determining whether $N$ accepts $w$ is equivalent to the problem of determining whether an accepting tableau for $N$ on $w$ exists.

Now we must give an $f$ to reduce $A$ to $SAT$. 
**Goal:**
On input $w$, the reduction produces a formula $\phi$.

**Variables of $\phi$:**
- Say that $Q$ and $\Gamma$ are the state set and tape alphabet of $N$, respectively.
- Let $C = Q \cup \Gamma \cup \{\#\}$.
- For each $i$ and $j$ between 1 and $n^k$ and for each $s$ in $C$, we have a variable, $x_{i,j,s}$.
- Each of the $(n^k)^2$ entries of a tableau is called a cell. The cell in row $i$ and column $j$ is called $\text{cell}[i,j]$ and contains a symbol from $C$.
- We represent the contents of the cells with the variables of $\phi$. If $x_{i,j,s}$ takes on the value 1, it means that $\text{cell}[i,j]$ contains an $s$. 
CONTINUING IDEA:
Design \( \varphi \) so that a satisfying assignment to the variables does correspond to an accepting tableau for \( N \) on \( w \).

The formula \( \varphi \) is the AND of four parts:
\[
\varphi_{\text{cell}} \land \varphi_{\text{start}} \land \varphi_{\text{move}} \land \varphi_{\text{accept}}
\]

Remember: turning variable \( x_{i,j,s} \) on corresponds to placing symbol \( s \) in cell \([i, j]\).

Therefore, must guarantee that the assignment turns on exactly one variable for each cell.

\[
\varphi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s, t \in C \atop s \neq t} (x_{i,j,s} \lor \overline{x_{i,j,t}}) \right) \right]
\]

\( \lor \) is shorthand for \( x_{i,j,s_1} \lor x_{i,j,s_2} \lor \cdots \lor x_{i,j,s_l} \)

INTUITION:
\( \varphi_{\text{cell}} \) merely says that, for every cell in the tableau \((i \text{ and } j)\), there is exactly one value in \( C \) which will result in the cell being a 1 (or true).

This is the first part of \( \varphi \).
Cook-Levin Theorem (cont.)

φ\text{start} = x_{1,1},\# \land x_{1,2},q_0 \land \ldots \land x_{1,n+2},w_n \land \ldots \land x_{1,n+3,\sqcup} \land \ldots \land x_{1,n^k-1,\sqcup} \land x_{1,n^k,\#}.

φ\text{start} ensures that the first configuration is the correct starting configuration.

φ\text{move} guarantees that each row of the tableau corresponds to a configuration that legally follows the preceding row’s configuration according to N’s rules. (Remember PCP?)

It does so by ensuring that each 2 × 3 window of cells is legal (e.g., configuration follows from the previous).

Ex: \(\delta(q_1, a) = \{(q_1, b, R)\}\)
\(\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}\).

φ\text{accept} ensures that an accept state appears somewhere in the tableau.

Valid windows:

\[
\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}}.
\]
Cook-Levin Theorem (cont.)

\[ \phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} (\text{the } (i, j)-\text{window is legal}) \]

The text “the \((i, j)-\text{window is legal}\)” may be replaced with the following formula:

\[ \bigvee_{a_1, \ldots, a_6 \text{ is a legal window}} (x_{i,j-1,a_1} \land x_{i,j,a_2} \land x_{i,j+1,a_3} \land x_{i+1,j-1,a_4} \land x_{i+1,j,a_5} \land x_{i+1,j+1,a_6}) \]

Now we must prove that the reduction runs in \textit{polynomial time}.

First, estimate the number of \textit{variables} in \(\phi\).

Tableau is a \(n^k \times n^k\) table, so it contains \(n^{2k}\) cells.

Each cell has \(l\) variables associated with it, where \(l\) is the number of symbols in \(C\).

Because \(l\) depends only on the TM \(N\) (\(N\) decides \(A\) in \(n^k\) time) and not on the length of the input \(n\), the total number of variables is \(O(n^{2k})\).

Now to estimate the size of each of the parts of \(\phi\).
Cook-Levin Theorem (cont.)

Formula $\phi_{\text{cell}}$ contains a fixed-size fragment of the formula for each cell of the tableau, so its size is $O(n^{2k})$.

Formula $\phi_{\text{start}}$ has a fragment for each cell in the top row, so its size is $O(n^k)$.

Formulas $\phi_{\text{move}}$ and $\phi_{\text{accept}}$ each contain a fixed-size fragment of the formula for each cell of the tableau, so their size is $O(n^{2k})$.

Thus, $\varphi$’s total size is $O(n^{2k})$, which is polynomial.

Cook-Levin Theorem is complete, because it is shown that:

1. SAT is in NP.
2. All other problems in NP can be reduced to SAT in polynomial time.

Therefore, SAT is proved to be NP-Complete.

Almost no other problem will take this much work.

Is it obvious to you that 3-SAT is NP-Complete?
P and NP Topics
