what is difficult?

- depends on the intelligence
  - human age
  - training
- depends on the degree of smartness
  - fast computers
  - size of background memory
- depends on viewpoint
  - how long does it takes
  - exact answer is needed

what is difficult for a computer?

what is a computer?
finite state machines

a finite state machine has

- $k$ states (one is the initial state)
- $m$ inputs
- $n$ outputs
- transition rules for each state and input
- output rules for each state
can fsm's solve all common problems?

NO, there exist common problems that cannot be "effectively computed" by finite state machines!!!

for example: multiplying two arbitrary binary numbers
doubling a sequence of "1"s
checking for palindromes
checking for balanced parenthesis

**problem:**
for arbitrarily long paren sequences, arbitrarily many states are required

- okay
- no good!
- no good!

A finite state machine can only keep track of a finite number of objects.
unbounded-space computation

during 1920s and 30s, much of the "science" part of computer science was being developed (long before actual electronic computers existed).

many different "models of computation" were proposed, and the classes of "functions" which could be computed by each were analyzed.

one of these models was the turing machine named after Alan Turing.

A "turing machine" is just a finite state machine which receives its inputs from and writes outputs onto a tape with unlimited extendability ..... solving "finiteness" problem of finite state machines.
turing machine specification

- a tape
  - discrete symbol positions
  - that can be extended at both sides (without bound)
- finite alphabet - say \{0, 1\}
- finite state machine control
  inputs:
  - symbol under the reading head
  outputs:
  - write 0/1
  - move left/right
- initial starting state \{S_0\}
- halt state \{Halt\}
a turing parity counter

the transition diagram

find the parity of the symbols between the head and the symbol "E", and replace that symbol by the parity

<table>
<thead>
<tr>
<th>current state</th>
<th>read</th>
<th>next state</th>
<th>write</th>
<th>direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>even</td>
<td>0</td>
<td>even</td>
<td>0</td>
<td>right</td>
</tr>
<tr>
<td>even</td>
<td>1</td>
<td>odd</td>
<td>0</td>
<td>right</td>
</tr>
<tr>
<td>odd</td>
<td>0</td>
<td>odd</td>
<td>0</td>
<td>right</td>
</tr>
<tr>
<td>odd</td>
<td>1</td>
<td>even</td>
<td>0</td>
<td>right</td>
</tr>
<tr>
<td>even</td>
<td>E</td>
<td>halt</td>
<td>0</td>
<td>stop</td>
</tr>
<tr>
<td>odd</td>
<td>E</td>
<td>halt</td>
<td>1</td>
<td>stop</td>
</tr>
</tbody>
</table>
a turing parenthesis checker

```
q0

O E ( ( ) ) ( ) ( ( ) ) ) ( ) E O O
```

```
q0

O E ( ( ) ) ( ) ( ( ) ) ( ) E O O
```

```
q0

O E ( ( ) ) ( ) ( ( ) ) ) ( ) E O O
```

```
q0

O E ( ( ) ) ( ) ( ( ) ) ( ) E O O
```

```
q1

O E ( ( X ) ) ( ) ( ( ) ) ) ( ) E O O
```

```
q0

O E ( ( X X ) ) ( ) ( ( ) ) ) ( ) E O O
```

next sheet
a turing parenthesis checker

etc., etc.
a turing parenthesis checker

this shows the superiority of turing machines over finite state machines

<table>
<thead>
<tr>
<th>current state</th>
<th>read</th>
<th>next state</th>
<th>write</th>
<th>direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>)</td>
<td>q1</td>
<td>X</td>
<td>left</td>
</tr>
<tr>
<td>q0</td>
<td>(</td>
<td>q0</td>
<td>X</td>
<td>left</td>
</tr>
<tr>
<td>q0</td>
<td>E</td>
<td>q2</td>
<td>E</td>
<td>right</td>
</tr>
<tr>
<td>q0</td>
<td>X</td>
<td>q0</td>
<td>X</td>
<td>right</td>
</tr>
<tr>
<td>q1</td>
<td>)</td>
<td>q1</td>
<td>X</td>
<td>left</td>
</tr>
<tr>
<td>q1</td>
<td>(</td>
<td>q0</td>
<td>X</td>
<td>left</td>
</tr>
<tr>
<td>q1</td>
<td>E</td>
<td>q1</td>
<td>X</td>
<td>left</td>
</tr>
<tr>
<td>q1</td>
<td>X</td>
<td>q0</td>
<td>X</td>
<td>right</td>
</tr>
<tr>
<td>q2</td>
<td>)</td>
<td>q2</td>
<td>0</td>
<td>stop</td>
</tr>
<tr>
<td>q2</td>
<td>(</td>
<td>q2</td>
<td>1</td>
<td>stop</td>
</tr>
<tr>
<td>q2</td>
<td>E</td>
<td>q2</td>
<td>X</td>
<td>left</td>
</tr>
</tbody>
</table>

remarks:
- the next state and direction are 1-1
- in "q2" the situation with ")" will never occur

not exactly a moore machine, but almost ....
what is a computer?

- a finite state machine
- instructions storing the algorithm \( A \)
- a program counter \( pc \) keeping track of where we are \( A \)
- additional state information to control \( A \)
- data stored to fix the instance to be solved
- some workspace for scratch data
- anything can be accessed and written in constant time
what is difficult for a computer?

- most famous "undecidable" problem: the halting problem

```plaintext
while x is not 1 do
  if x is even do x := x/2
  else do x := 3x + 1
```

- chip library problem (Wang's dominoes)
chip library problem
DNA self-assembly problems
what is computable?

• anything computable, can be computed by simple machines
• we do not even have such a simple machine
• anything inefficient on such a machine, is inefficient on any reasonable machine
• let us pretend that our finite machine has no memory size limitations
what is difficult for a computer?

• not computable (undecidable)
• if computable, what does it take?
  - how much time?
  - how much memory space?
  - how many processors?

computer science answer: complexity classes
class structures

a problem is efficiently solvable if there exists a polynomial time algorithm for it

- P
- Log Time
- Log Space
- Pspace
- ExpTime
- ExpSpace

- sorting
- linear programming
- shortest paths

- searching
efficiency?

polynomial-time algorithm: on inputs of size $n$, it runs in $O(n^k)$ for some constant $k$

<table>
<thead>
<tr>
<th>complexity</th>
<th>now</th>
</tr>
</thead>
<tbody>
<tr>
<td>speedup</td>
<td>$1\times$</td>
</tr>
<tr>
<td>linear</td>
<td>X</td>
</tr>
<tr>
<td>quadratic</td>
<td>Y</td>
</tr>
<tr>
<td>exponential</td>
<td>Z</td>
</tr>
</tbody>
</table>
A printed circuit board, or PCB, is used to mechanically support and electrically connect electronic components using conductive pathways, tracks, or traces, etched from copper sheets laminated onto a non-conductive substrate. Holes are drilled in the board so that components can be inserted and via's realized.

The drilling is performed by automated drilling machines with placement controlled by a drill tape or drill file. The drill file describes the location and size of each drilled hole. These holes are often filled with annular rings (hollow rivets) to create via's.
printed circuit board drilling

time to drill a board = number of holes x drill action per hole + sum of distances between subsequent holes

minimizing distance by generating all permutations of holes, do the summation and keep track of the smallest result, takes \( n! \) summations and comparisons.

Stirling's approximation, \( n! \approx n^n e^{-n} \sqrt{2\pi n} \), shows that that is at least of exponential time complexity.

Is there an essentially better (more efficient) algorithm?

However, it is easy (polynomial-time) to check whether a given sequence keeps the drilling time below a given time.
....sequential solution?

Suppose we have a decision problem for instance \( x \), proposed satisfying solution \( y \) and a checking algorithm \( A \).

For example: \( x \) is the drill file, \( y \) is the sequence of holes and time upper bound, \( A \) is the summation of subsequent hole distances and comparison with upper bound.
circuit-satisfiability NP-hard?

to prove: $\forall L \in \mathbf{NP} \ [ L \leq_p \text{circuit-satisfiability} ]$

take $L \in \mathbf{NP}$; find a polynomial-time computable $f: \{0,1\}^* \to \{0,1\}^*$ such that $x \in L$ if $f(x) \in \text{circuit-satisfiability}$ ($\forall x \in \{0,1\}^*$)

proof (sketch)

there is a verification algorithm $A$ for $L$

computer representation of $A(x,y)$:
and a combinatorial solution?
some intermediate conclusions

- polynomial number of repetitions of the combinatorial network $M$
- the result for the same input $y$ is the same output
- the total network is a combinatorial network with one output
- verification comes down to the question: is there an input $y$ that yields an output 1
- in general, this is called the circuit-satisfiability problem
circuit-satisfiability

- **given**: a boolean combinational circuit of AND, OR and NOT gates with one output
- **question**: does it have inputs such that the output becomes high?
circuit-satisfiability

• **given:** a boolean combinational circuit of AND, OR and NOT gates with one output
• **question:** does it have inputs such that the output becomes high?

![Circuit Diagram]

not satisfiable !!

$2^3$ possible input vectors!
class structures

class of problems of which can easily check decision correctness (i.e. in polynomial time)

problems that can be efficiently solved (i.e. in polynomial time)
circuit-satisfiability in \( \mathbf{NP} \)?

**input:** standard encoding of circuit \( C \), \( \langle C \rangle \)

**certificate:** bit string corresponding to truth assignment, \( O \mid \langle C \rangle \mid \)

**algorithm:**
- sort topologically
- simulate

clearly polynomial-time

\[ \text{circuit-satisfiability in \( \mathbf{NP} \)} \]
which problems are in NP?

suppose we have a decision problem for instance $x$, proposed satisfying solution $y$ (with a non-exploding length) and an efficient checking algorithm $A$
and combinatorially?
polynomial-time reducibility and $P$

Theorem: when problem 2 can be decided in polynomial time, and every instance of problem 1 can be converted into an instance of problem 2!

Proof:

- $x$ is input
- $f$ is a polynomial-time function
- $f(x)$ is output
- $A_2$ is a polynomial-time algorithm
- $f(x)$ is decided
- $x$ is decided
polynomial-time reducibility and P

Theorem: when problem A2 can be decided in polynomial time, and every instance of problem A1 can be converted into an instance of problem A2 in polynomial time, then problem A1 can be decided in polynomial time!

Proof:

\[ \{0,1\}^* \xrightarrow{f(x)} \{0,1\}^* \]

\[ A_1 \xrightarrow{f(x)} A_2 \]

\[ f(x) \text{ decided} \rightarrow x \text{ decided} \]
L₁ reducible to L₂ if there is an f: {0,1}∗ → {0,1}∗ such that
x in L₁ if f(x) in L₂ (∀ x in {0,1}*)

L₁ polynomial-time reducible to L₂ if f polynomial-time computable
notation: L₁ ≤ₚ L₂

L₁ cannot be more difficult than L₂
example: sets and covers

-independent sets-

\{3, 4, 5\} and \{1, 4, 5, 6\}

-vertex covers-

\{1, 2, 6, 7\} and \{2, 3, 7\}

S is an IS if V\(\setminus S\) is an VC

IS \(\leq_p\) VC
VC \(\leq_p\) IS
VC \(\leq_p\) SC

Given a set \(U\), a collection subsets of \(U\), and an integer \(k\), is there a collection of at most \(k\) subsets whose union is \(U\)?
circuit-satisfiability the hardest in NP?

\[
\begin{align*}
\alpha(1) & \quad \alpha(1) \quad \alpha(n^2) \quad |x|=n \quad \vdots \\
c_1 & \quad A \quad pc \quad state \quad x \quad y \quad workspace \\
M & \quad \alpha(n^3) \\
M & \quad \alpha(n^4) \\
c_2 & \quad A \quad pc \quad state \quad x \quad y \quad workspace \\
M & \quad \alpha(n^3) \\
\vdots & \quad \vdots \\
M & \\
M & \\
\alpha(n^m) & \quad A \quad pc \quad state \quad x \quad y \quad workspace \quad output \\
circuit f(x) & \quad \text{deciding } x \quad \iff \quad f(x) \text{ satisfiable} \\
f \text{ pol-time comp} & \\
\text{circuit-satisfiability} & \quad \text{the hardest} \\
\text{circuit-satisfiability} & \quad \text{NP-complete}
\end{align*}
\]
formula-satisfiability (SAT)

given: a well-formed boolean expression
question: is it satisfiable?
   (does it have a truth assignment such that it evaluates to 1?)

1. SAT in NP
2. reduction from circuit-satisfiability

3. inductive substitution doesn't work!
   (not polynomial-time computable)

   \[ x_9 \leftrightarrow (x_6 \land x_7 \land x_8) \]
formula-satisfiability (SAT)

given: a well-formed boolean expression

question: is it satisfiable?

(does it have a truth assignment such that it evaluates to 1?)

1. SAT in NP
2. reduction from circuit-satisfiability

3. inductive substitution does not work!
   (not polynomial-time computable)

   \[ \phi = x_9 \land (x_9 \leftrightarrow (x_6 \land x_7 \land x_8)) \land (x_8 \leftrightarrow (x_3 \lor x_6)) \land (x_7 \leftrightarrow (x_3 \lor x_5)) \land (x_6 \leftrightarrow (x_1 \land x_2 \land x_4)) \land (x_5 \leftrightarrow (x_1 \lor x_2)) \land (x_4 \leftrightarrow \neg x_3) \]

4. circuit satisfiable if formula satisfiable
5. mapping polynomial-time computable  SAT in NPC
3-CNF-SAT

given: A boolean expression that is a product of 3-OR clauses
question: Is it satisfiable? \((\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor \neg x_4)\)

1. 3-CNF-SAT in NP

2. reduction from SAT

3. \(\phi = ((x_1 \rightarrow x_2) \lor \neg((\neg x_1 \leftrightarrow x_3) \lor x_4)) \land \neg x_2\)

\(\psi = y_1 \land (y_1 \leftrightarrow (y_2 \land \neg x_2)) \land (y_2 \leftrightarrow (y_3 \lor 44)) \land (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \land (y_4 \leftrightarrow \neg y_5) \land (y_5 \leftrightarrow (y_6 \lor x_4)) \land (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))\)

\(\psi_1 = (\neg y_1 \lor \neg y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor x_2) \land (y_1 \lor \neg y_2 \lor x_2)\)

\(\psi_4 = (\neg y_4 \lor \neg y_5) \land (y_4 \lor y_5)\)

4. \(\phi\) satisfiable if \(f(\phi)\) satisfiable

5. mapping polynomial-time computable

3-CNF-SAT in NPC
clique

given: an undirected graph \( G=(V,E) \) and a number \( k \)

question: does \( G \) have a clique (a complete subgraph) of size \( k \) (vertices)?

1. CLIQUE in \( \mathbf{NP} \)
2. reduction from 3-CNF-SAT
3. \( \phi = (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_3) \) (with \( k=3 \) clauses)

4. assume \( \phi \) satisfiable
   \( \Rightarrow \) every clause has \( \geq 1 \) true literal
   \( \Rightarrow \) choose corresponding nodes
   \( \Rightarrow \) clique of size 3
   assume clique of size 3
   \( \Rightarrow \) make corresponding literals true
   \( \Rightarrow \) satisfying assignment
   \( \phi \) satisfiable if \( f(\phi) \) has clique of size \( k \)

5. mapping polynomial-time computable

\( \text{CLIQUE in} \ \mathbf{NPC} \)
vertex cover (VC)

given: an undirected graph $G=(V,E)$ and a number $k$

vertex cover: subset $U$ of $V$ such that for any $(u,v)$ in $E$, $u$ in $U$ or $v$ in $U$

question: does $G$ have a vertex cover of size $k$?

1. VC in $\text{NP}$
2. reduction from CLIQUE
3. CLIQUE instance: complement:

4. $G$ has a clique of size $k$ if f($G$) has a vertex cover of size $|V|-k$
5. mapping polynomial-time computable

VC in $\text{NPC}$
subset sum

given: a set of natural numbers \( S \) and a target number \( t \)

question: does \( S \) have a subset \( R \) that sums to \( t \)?

1. subset sum in \( \text{NP} \)
2. reduction from \( \text{VC} \)
3. \( G, k(=3) \)

incidence matrix:

\[
\begin{array}{ccccc}
& e_4 & e_3 & e_2 & e_1 & e_0 \\
\text{v0} & 0 & 0 & 1 & 0 & 1 \\
\text{v1} & 1 & 0 & 0 & 1 & 0 \\
\text{v2} & 1 & 1 & 0 & 0 & 0 \\
\text{v3} & 0 & 0 & 1 & 0 & 0 \\
\text{v4} & 0 & 1 & 0 & 1 & 1 \\
\end{array}
\]

modified base 4 numbers

\[
\begin{array}{cccccc}
\text{x0} & 1 & 0 & 0 & 1 & 0 & 1 & = 1041 \\
\text{x1} & 1 & 1 & 0 & 0 & 1 & 0 & = 1284 \\
\text{x2} & 1 & 1 & 1 & 0 & 0 & 0 & = 1344 \\
\text{x3} & 1 & 0 & 0 & 1 & 0 & 0 & = 1040 \\
\text{x4} & 1 & 0 & 1 & 0 & 1 & 1 & = 1093 \\
\text{y0} & 0 & 0 & 0 & 0 & 0 & 1 & = 1 \\
\text{y1} & 0 & 0 & 0 & 0 & 1 & 0 & = 4 \\
\text{y2} & 0 & 0 & 0 & 1 & 0 & 0 & = 16 \\
\text{y3} & 0 & 0 & 1 & 0 & 0 & 0 & = 64 \\
\text{y4} & 0 & 1 & 0 & 0 & 0 & 0 & = 256 \\
\end{array}
\]

4. \( G \) has a \( \text{VC} \) of size \( k \) if \( f(G) \) has a subset with sum \( t \)

5. mapping polynomial-time computable

subset sum in \( \text{NPC} \)
**hamilton circuit**

**given:** an undirected graph \( G=(V,E) \)

**question:** does \( G \) have a **hamilton circuit**
(a circuit going through all vertices)?

1. **HAMCIRCUIT** in **NP** (certify a given sequence of vertices!)
2. reduction from **3-CNF-SAT**
3. convert any boolean product of or-clauses with 3 literals into an undirected graph, making sure that
   - for each boolean variable there is **choice** between two values
   - all occurrences of a variable \( x \) get the same value (**consistency**)
   - encode the **constraints** imposed by the clauses

three gadgets, one for each requirement, **only** connected to the rest of the graph through end vertices (full dots):

- **logic gadget**
- **consistency gadget**
- **constraint gadget**

denoted by:
hamilton circuit

properties of the gadgets:

logic gadgets: one for each variable

entering through one end vertex, the circuit has to leave through the other one, using only one of the two internal edges: fixing the value of the variable

consistency gadgets: one for each literal

can only be traversed in only one of two ways!

we need an even number of crossings larger than two!

constraint gadgets:

one for each clause

if a side is traversed only if the corresponding literal is false, at least one of the literals in a clause has to be true!

make a corresponding choice between the logic edges to include!
\[ \phi = (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3) \]

4. the graph has hamiltonian circuit if and only if \( \phi \) has a satisfying truth assignment for the variables

5. mapping polynomial-time computable
traveling salesman circuit

given: a finite set of n cities, a positive integer distance between each pair, and a positive integer B

question: is there a permutation \( \pi \) of the \( \{1, 2, \ldots, n\} \) such that

\[
\sum_{i=1}^{n-1} d(\pi(i), \pi(i+1)) + d(\pi(1)\pi(n)) \leq B
\]

1. TSC in \( \text{NP} \) (certify a given sequence of cities!)

2. reduction from \( \text{HAM-CIRC} \)

   given: an undirected graph \( G=(V,E) \)
   - for each vertex create a city
   - for each city pair set the distance to 1 if there is an edge between the corresponding vertices else make the distance \(|V|+1\)
   - set \( B = |V| \)

4. if \( G \) has an hamiltonian circuit, then there is a route of length \( B \) (namely that circuit which is a subgraph of the complete graph); if there is a route of length \( B \), then this route can only use distances of length 1, which corresponds to the edges of the given graph!

5. mapping polynomial-time computable
NP-complete problems

CIRCUIT-SAT

SAT

3-CNF-SAT

CLIQUE

VC

SUBSET-SUM

HAM-CIRC

TSC