Content of the Course

Foundations of Computing II

Sven Seuken and Dennis Komm

September 14, 2020

Goal

The central questions are

- What **cannot** be done using a computer?
- What **cannot** be done efficiently using a computer?

Problems in Math

Problems in Math

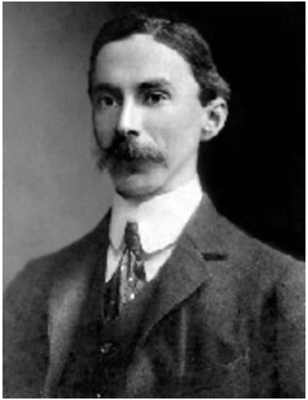
Problems in Math

Gottfried Wilhelm Leibniz (1646 – 1716)

Wikimedia, Creative Commons

"Let us calculate, without further ado, to see who is right."

Bertrand Russell (1872 – 1970)



Wikimedia, Creative Commons

- Let S be the set containing all sets that do not contain themselves, that is, $S = \{X \mid X \text{ is a set and } X \notin X\}$
- Does S contain itself, that is, $S \in S$ or $S \notin S$?
- This leads to a contradiction
- As a consequence, S cannot exist

Bertrand Russell (1872 – 1970)

Suppose we have $S \in S$

Bertrand Russell (1872 – 1970)

Suppose we have $S \in S$

- ⇨ Thus, S is a set that contains itself
- ⇨ But then S cannot be in S, because S is defined such that it does not contain any set that contains itself

Bertrand Russell (1872 – 1970)

Suppose we have $S \in S$

- ⇨ Thus, S is a set that contains itself
- ⇨ But then S cannot be in S, because S is defined such that it does not contain any set that contains itself

Oh well, then suppose we have $S \notin S$

Bertrand Russell (1872 – 1970)

Suppose we have $S \in S$

- ⇨ Thus, S is a set that contains itself
- ⇨ But then S cannot be in S, because S is defined such that it does not contain any set that contains itself

Oh well, then suppose we have $S \notin S$

- ⇨ Ok, so S is a set that does not contain itself
- ⇨ But then S must be in S, because that is exactly its definition

Suppose we have $S \in S$

⇨ Thus, S is a set that contains itself

⇨ But then S cannot be in S, because S is defined such that it does not contain any set that contains itself

Oh well, then suppose we have $S \notin S$

⇨ Ok, so S is a set that does not contain itself

⇨ But then S must be in S, because that is exactly its definition

Contradiction

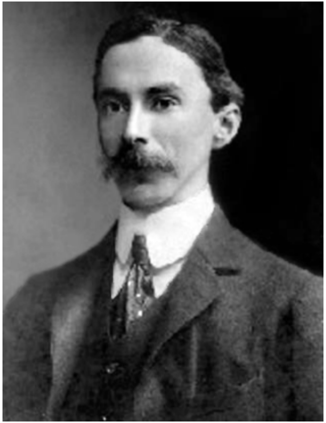
David Hilbert (1862 – 1943)

Wikimedia, Creative Commons

Hilbert Program

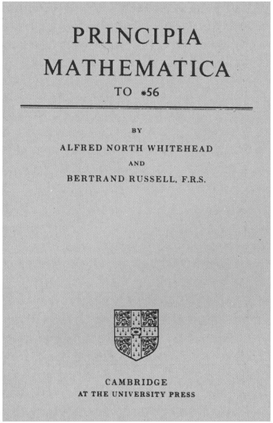
Create a system of axioms in which every true statement can be proven, and that does not contain any paradoxes and inconsistencies

B. Russell, Alfred N. Whitehead (1861 – 1947)



Wikimedia, Creative Commons

Unknown



Wikimedia, Creative Commons

Kurt Gödel (1862 – 1943)

Wikimedia, Creative Commons

Gödel's Incompleteness Theorem

If a system is consistent (there are no paradoxes) and powerful enough, then there are true statements in this system that cannot be proven (it is incomplete)

Conclusion

Today we know (thanks to Gödel) that ...

- Mathematics is incomplete
- There are statements that we can neither prove nor disprove
- This will stay that way

Conclusion

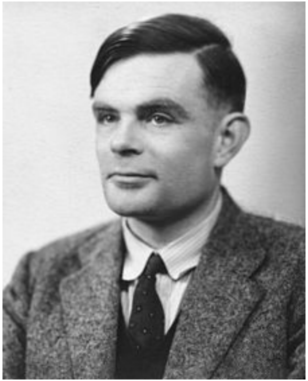
Today we know (thanks to Gödel) that . . .

- Mathematics is incomplete
- There are statements that we can neither prove nor disprove
- This will stay that way

Around the same time, an English mathematician asked similar questions

- What can be automated?
- Where lies the border to what cannot be automated?

Alan M. Turing (1912 – 1954)



Wikimedia, Creative Commons

There are provable statements that we cannot prove automatically (that is, we cannot compute the answer)

The idea is very similar to the one of Gödel's Incompleteness Theorem

Infinity times Infinity

Infinity times Infinity

Infinity times Infinity

Walt Disney Pictures

Infinity times Infinity

- A shepherd has black and white sheep
- Does he have more black ones or more white ones?
- He can only count to three

Infinity times Infinity

- A shepherd has black and white sheep
- Does he have more black ones or more white ones?
- He can only count to three

⇨ Pair one black sheep with a white sheep and bring them to another meadow

Juraj Hromkovič, *Berechenbarkeit*, Vieweg Teubner

Infinity times Infinity

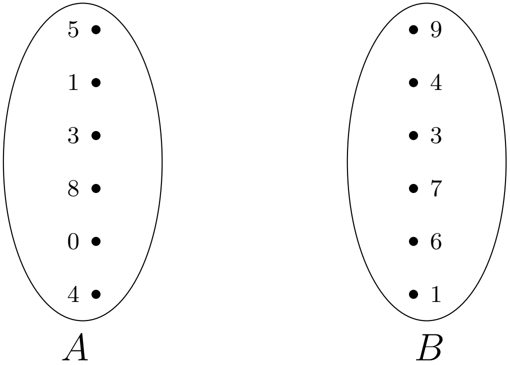
Theorem (of Cantor and Bernstein)

Two sets A and B have the same size if there is a bijection between them $\square$

Infinity times Infinity

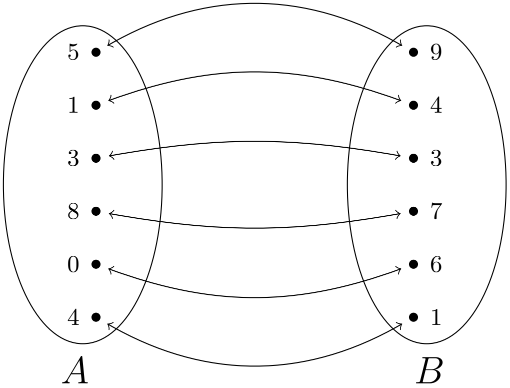
Theorem (of Cantor and Bernstein)

Two sets A and B have the same size if there is a bijection between them □



Theorem (of Cantor and Bernstein)

Two sets A and B have the same size if there is a bijection between them □



Infinity times Infinity

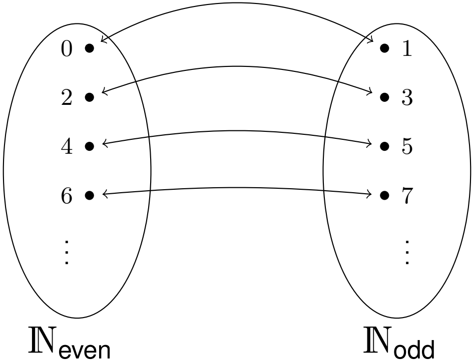
This is also true for infinite sets

- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ (natural numbers)
- $\mathbb{N}_{\text{even}} = \{0, 2, 4, 6, \dots\}$ (even numbers)
- $\mathbb{N}_{\text{odd}} = \{1, 3, 5, 7, \dots\}$ (odd numbers)

Infinity times Infinity

This is also true for infinite sets

- $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ (natural numbers)
- $\mathbb{N}_{\text{even}} = \{0, 2, 4, 6, \ldots\}$ (even numbers)
- $\mathbb{N}_{\text{odd}} = \{1, 3, 5, 7, \ldots\}$ (odd numbers)



Infinity times Infinity

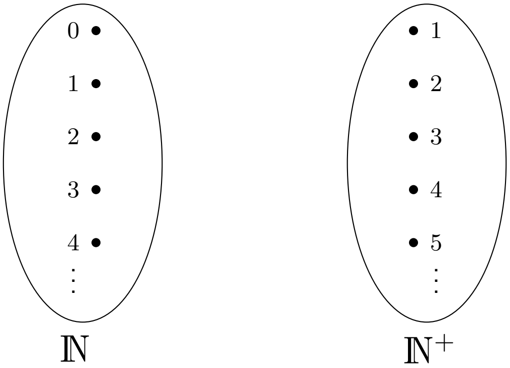
⇨ Thus we have $|\mathbb{N}_{\text{even}}| = |\mathbb{N}_{\text{odd}}|$

- How about $\mathbb{N}^+ = \{1, 2, 3, \dots\}$? (positive natural numbers)
- Intuitively, $|\mathbb{N}^+| < |\mathbb{N}|$ should be true

Infinity times Infinity

⇨ Thus we have $|\mathbb{N}_{\text{even}}| = |\mathbb{N}_{\text{odd}}|$

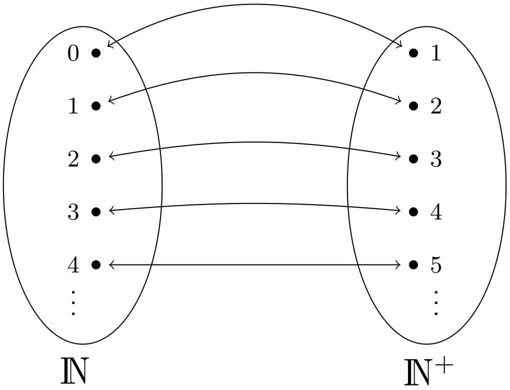
- How about $\mathbb{N}^+ = \{1, 2, 3, \dots\}$? (positive natural numbers)
- Intuitively, $|\mathbb{N}^+| < |\mathbb{N}|$ should be true



Infinity times Infinity

⇨ Thus we have $|\mathbb{N}_{\text{even}}| = |\mathbb{N}_{\text{odd}}|$

- How about $\mathbb{N}^+ = \{1, 2, 3, \dots\}$? (positive natural numbers)
- ~~Intuitively, $|\mathbb{N}^+| < |\mathbb{N}|$ should be true~~; we have $|\mathbb{N}^+| = |\mathbb{N}|$



Infinity times Infinity

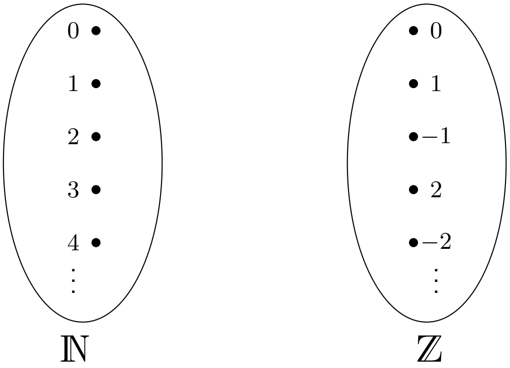
⇨ Ok, so we also have $|\mathbb{N}^+| = |\mathbb{N}|$

- How about $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$? (integers)
- Intuitively, $|\mathbb{Z}| = 2 \cdot |\mathbb{N}|$ should be true

Infinity times Infinity

⇨ Ok, so we also have $|\mathbb{N}^+| = |\mathbb{N}|$

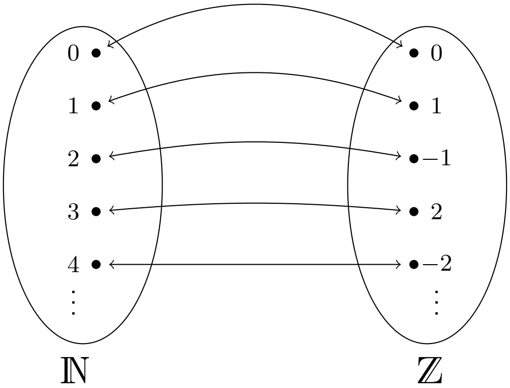
- How about $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$? (integers)
- Intuitively, $|\mathbb{Z}| = 2 \cdot |\mathbb{N}|$ should be true



Infinity times Infinity

⇨ Ok, so we also have $|\mathbb{N}^+| = |\mathbb{N}|$

- How about $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$? (integers)
- ~~Intuitively, $|\mathbb{Z}| = 2 \cdot |\mathbb{N}|$ should be true~~; we have $|\mathbb{Z}| = |\mathbb{N}|$



Infinity times Infinity

Definition

An infinite set A is called **countable** if it has the same size as $\mathbb{N}$

Infinity times Infinity

Definition

An infinite set A is called **countable** if it has the same size as $\mathbb{N}$

- ⇨ Every element of A is assigned to a natural number
- ⇨ We can **enumerate** (count) the elements of A

Infinity times Infinity

Definition

An infinite set A is called **countable** if it has the same size as $\mathbb{N}$

- ⇨ Every element of A is assigned to a natural number
- ⇨ We can **enumerate** (count) the elements of A

Theorem

$\mathbb{Z}$ is countable

Infinity times Infinity

Definition

An infinite set A is called **countable** if it has the same size as $\mathbb{N}$

⇨ Every element of A is assigned to a natural number

⇨ We can **enumerate** (count) the elements of A

Theorem

$\mathbb{Z}$ is countable

- $\mathbb{Q}^+ = \left\{\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \ldots\right\}$ (positive rational numbers)
- $\mathbb{Q} \ = \left\{\ldots, -\frac{1}{3}, -\frac{1}{2}, -\frac{1}{1}, \frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots\right\}$ (rational numbers)

Infinity times Infinity

Definition

An infinite set A is called **countable** if it has the same size as $\mathbb{N}$

⇨ Every element of A is assigned to a natural number

⇨ We can **enumerate** (count) the elements of A

Theorem

$\mathbb{Z}$ is countable

- $\mathbb{Q}^+ = \left\{\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \ldots\right\}$ (positive rational numbers)
- $\mathbb{Q} \;\; = \left\{\ldots, -\frac{1}{3}, -\frac{1}{2}, -\frac{1}{1}, \frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots\right\}$ (rational numbers)

Theorem

$\mathbb{Q}^+$ and $\mathbb{Q}$ are both countable

David Hilbert (1862 – 1943)

Wikimedia, Creative Commons

The Hilbert Hotel

- Infinite number of rooms
- There is one guest in every room

David Hilbert (1862 – 1943)

Wikimedia, Creative Commons

The Hilbert Hotel

- Infinite number of rooms
- There is one guest in every room
- A new guest arrives

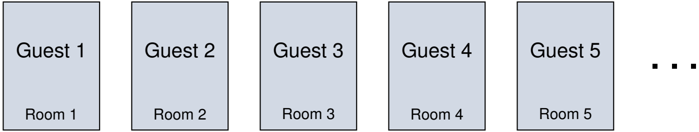
David Hilbert (1862 – 1943)

Wikimedia, Creative Commons

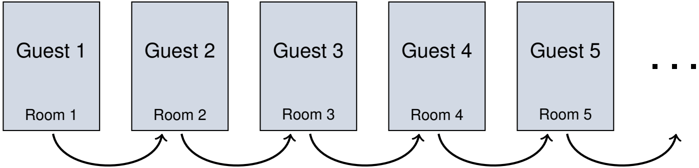
The Hilbert Hotel

- Infinite number of rooms
- There is one guest in every room
- A new guest arrives
- Can we assign this guest to a room without kicking anyone out?

A new guest arrives. . .

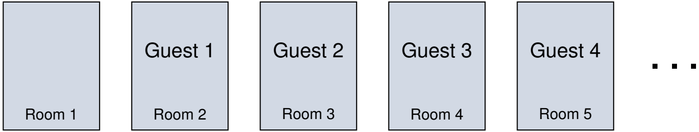


A new guest arrives...

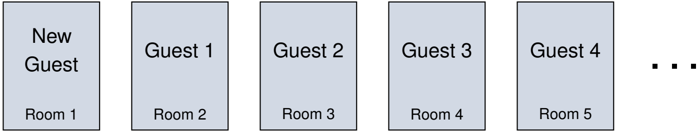


David Hilbert (1862 – 1943)

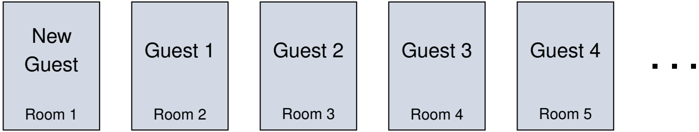
A new guest arrives...



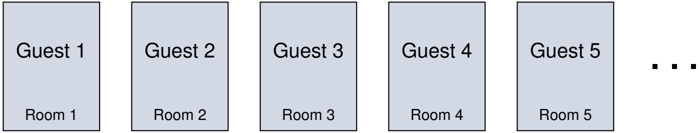
A new guest arrives...



A new guest arrives...

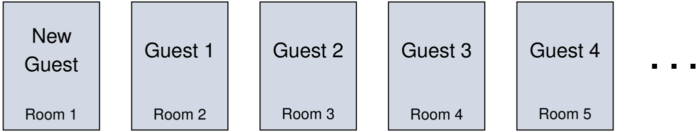


Infinitely many new guests arrive...

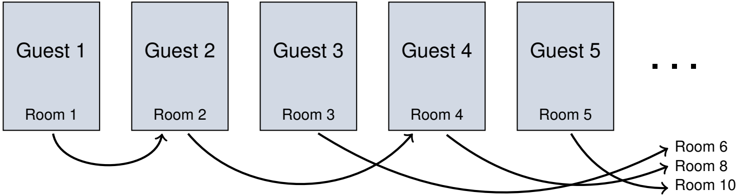


David Hilbert (1862 – 1943)

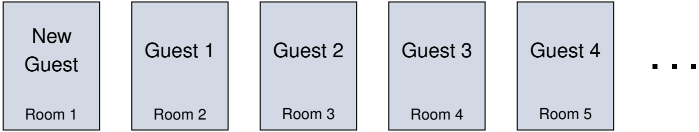
A new guest arrives. . .



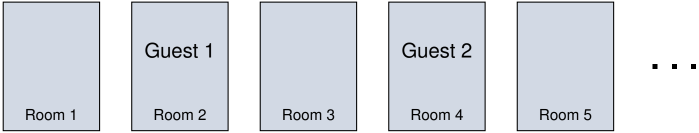
Infinitely many new guests arrive. . .



A new guest arrives…



Infinitely many new guests arrive…



David Hilbert (1862 – 1943)

A new guest arrives...

New Guest	Guest 1	Guest 2	Guest 3	Guest 4	...
Room 1	Room 2	Room 3	Room 4	Room 5	

Infinitely many new guests arrive...

New Guest 1	Guest 1	New Guest 2	Guest 2	New Guest 3	...
Room 1	Room 2	Room 3	Room 4	Room 5	

Uncountability of $\mathbb{R}$

- Are there sets that are actually larger than $\mathbb{N}$?
- Consider $\mathbb{R}$ (real numbers)
- Recall that $\pi, \sqrt{2} \in \mathbb{R}$, but $\pi, \sqrt{2} \notin \mathbb{Q}$

Uncountability of $\mathbb{R}$

- Are there sets that are actually larger than $\mathbb{N}$?
- Consider $\mathbb{R}$ (real numbers)
- Recall that $\pi, \sqrt{2} \in \mathbb{R}$, but $\pi, \sqrt{2} \notin \mathbb{Q}$

Theorem

$\mathbb{R}$ is not countable (we say **uncountable**)

Uncountability of $\mathbb{R}$

- Are there sets that are actually larger than $\mathbb{N}$?
- Consider $\mathbb{R}$ (real numbers)
- Recall that $\pi, \sqrt{2} \in \mathbb{R}$, but $\pi, \sqrt{2} \notin \mathbb{Q}$

Theorem

$\mathbb{R}$ is not countable (we say **uncountable**)

- Again, we prove the claim by **contradiction**
- Suppose the opposite were true (that is, "$\mathbb{R}$ is countable")
- We show that this leads to a contradiction

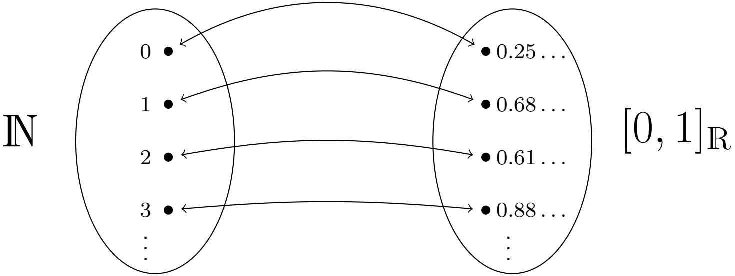
Uncountability of $\mathbb{R}$

- So suppose $\mathbb{R}$ were countable
- We even only look at the real numbers between 0 and 1, that is, $[0, 1]_{\mathbb{R}}$
- We suppose that we can enumerate (count) these numbers

Uncountability of $\mathbb{R}$

- So suppose $\mathbb{R}$ were countable
- We even only look at the real numbers between 0 and 1, that is, $[0,1]_{\mathbb{R}}$
- We suppose that we can enumerate (count) these numbers

⇨ Then we get a bijection:



Uncountability of $\mathbb{R}$

- So suppose $\mathbb{R}$ were countable
- We even only look at the real numbers between 0 and 1, that is, $[0,1]_{\mathbb{R}}$
- We suppose that we can enumerate (count) these numbers

⇨ Then we get a bijection, resp. a table as follows:

Number	Real number
0	$0.\ 2\ 5\ 6\ 5\ 1\ 4\ 0\ 5\ \ldots$
1	$0.\ 6\ 8\ 0\ 0\ 7\ 1\ 4\ 3$
2	$0.\ 6\ 1\ 7\ 3\ 9\ 0\ 1\ 9$
3	$0.\ 8\ 8\ 7\ 4\ 0\ 8\ 4\ 8$
$\vdots$	$\vdots \qquad \ddots$

Uncountability of $\mathbb{R}$

Number	Real number									
0	$0.$	2	5	6	5	1	4	0	5	
1	$0.$	6	8	0	0	7	1	4	3	$\dots$
2	$0.$	6	1	7	3	9	0	1	9	
3	$0.$	8	8	7	4	0	8	4	8	
$\vdots$				$\vdots$						$\ddots$

- Consider the ith decimal place of the ith number

Uncountability of $\mathbb{R}$

Number	Real number									
0	0.	$\boxed{2}$	5	6	5	1	4	0	5	
1	0.	6	$\boxed{8}$	0	0	7	1	4	3	$\dots$
2	0.	6	1	$\boxed{7}$	3	9	0	1	9	
3	0.	8	8	7	$\boxed{4}$	0	8	4	8	
$\vdots$				$\vdots$						$\ddots$

- Consider the ith decimal place of the ith number
- We construct a real number x
- x is different from the ith number at the ith decimal place
- Here, for instance $x = 0.1763\dots$

Uncountability of $\mathbb{R}$

Number	Real number									
0	$0.$	$\boxed{2}$	5	6	5	1	4	0	5	
1	$0.$	6	$\boxed{8}$	0	0	7	1	4	3	$\dots$
2	$0.$	6	1	$\boxed{7}$	3	9	0	1	9	
3	$0.$	8	8	7	$\boxed{4}$	0	8	4	8	
$\vdots$				$\vdots$						$\ddots$

- Consider the ith decimal place of the ith number
- We construct a real number x
- x is different from the ith number at the ith decimal place
- Here, for instance $x = 0.1763\dots$

⇨ x cannot be in the table

Uncountability of $\mathbb{R}$

Number	Real number									
0	$0.$	$\boxed{2}$	5	6	5	1	4	0	5	
1	$0.$	6	$\boxed{8}$	0	0	7	1	4	3	$\ldots$
2	$0.$	6	1	$\boxed{7}$	3	9	0	1	9	
3	$0.$	8	8	7	$\boxed{4}$	0	8	4	8	
$\vdots$				$\vdots$						$\ddots$

- Consider the ith decimal place of the ith number
- We construct a real number x
- x is different from the ith number at the ith decimal place
- Here, for instance $x = 0.1763\ldots$

⇨ x cannot be in the table

Contradiction

Uncountability of $\mathbb{R}$

Number	Real number									
0	0.	$\boxed{2}$	5	6	5	1	4	0	5	
1	0.	6	$\boxed{8}$	0	0	7	1	4	3	…
2	0.	6	1	$\boxed{7}$	3	9	0	1	9	
3	0.	8	8	7	$\boxed{4}$	0	8	4	8	
⋮				⋮						⋱

- Consider the ith decimal place of the ith number
- We construct a real number x
- x is different from the ith number at the ith decimal place
- Here, for instance $x = 0.1763\ldots$

⇨ x cannot be in the table

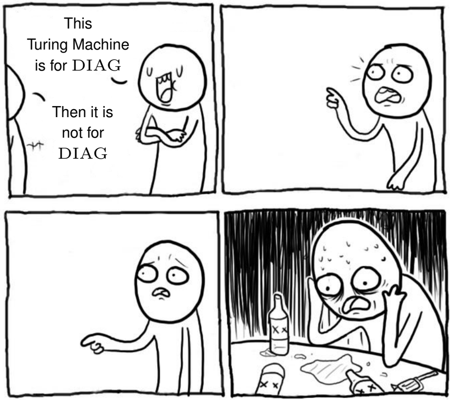
⇨ This method is called **diagonalization**

Contradiction

The Turing Machine

The Turing Machine

The Turing Machine



The Turing Machine

Now we want to show that there are problems that we cannot solve automatically (that is, algorithmically); the basic idea is...

- There is a countable number of algorithms
- But there is an uncountable number of problems

The Turing Machine

Now we want to show that there are problems that we cannot solve automatically (that is, algorithmically); the basic idea is...

- There is a countable number of algorithms
- But there is an uncountable number of problems

However, before that we need to meditate...

- What is a problem?
- What is an algorithm?

⇨ We need a precise mathematical definition

What is a Problem?

- An **alphabet** is a finite set of symbols (letters)
- A **word** is a string of these symbols
- $\mathrm{DEC} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is the decimal alphabet
- $x = 13817$ is a word “over” DEC

What is a Problem?

- An **alphabet** is a finite set of symbols (letters)
- A **word** is a string of these symbols
- $\mathrm{DEC} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is the decimal alphabet
- $x = 13817$ is a word "over" DEC

Definition

A **decision problem** L is to decide, for a given word x, whether it is in the set L, that is, whether $x \in L$ or $x \notin L$

What is a Problem?

- An **alphabet** is a finite set of symbols (letters)
- A **word** is a string of these symbols
- $\mathrm{DEC} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is the decimal alphabet
- $x = 13817$ is a word “over” DEC

Definition

A **decision problem** L is to decide, for a given word x, whether it is in the set L, that is, whether $x \in L$ or $x \notin L$

- The answer is always YES or NO
- As a matter of fact, this modelling is very general

What is a Problem?

- $\mathrm{PRIMES} = \{y \mid y \text{ is a prime number}\}$ (y is over DEC)
- Is x a prime number?

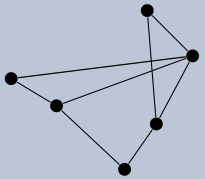
⇨ Is $x \in \mathrm{PRIMES}$?

What is a Problem?

- $\mathrm{PRIMES} = \{y \mid y \text{ is a prime number}\}$ (y is over DEC)
- Is x a prime number?

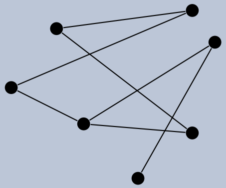
⇨ Is $x \in \mathrm{PRIMES}$?

- $\mathrm{HC} = \{y \mid y \text{ is a graph with a Hamiltonian cycle}\}$
- The graph



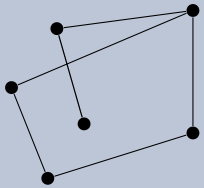
is in HC

- The graph



is not in HC

- The graph



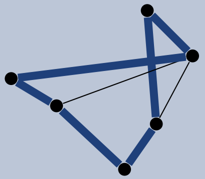
neither

What is a Problem?

- $\mathrm{PRIMES} = \{y \mid y$ is a prime number$\}$ (y is over DEC)
- Is x a prime number?

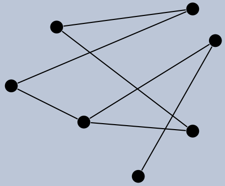
⇨ Is $x \in \mathrm{PRIMES}$?

- $\mathrm{HC} = \{y \mid y$ is a graph with a Hamiltonian cycle$\}$
- The graph



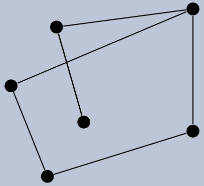
is in HC

- The graph



is not in HC

- The graph



neither

What is an Algorithm?

Wikimedia, Creative Commons

Definition

An **algorithm** is a finite step-by-step method to solve every instance of a given problem in finite time

What is an Algorithm?

Wikimedia, Creative Commons

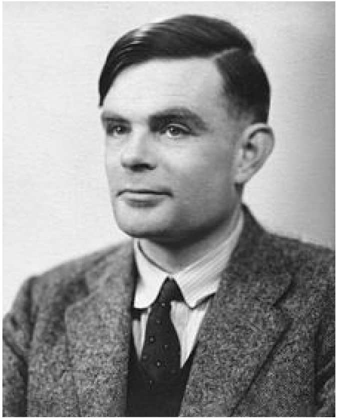
Definition

An **algorithm** is a finite step-by-step method to solve every instance of a given problem in finite time

- A computer program that checks whether a given number is a prime number
- Recipe, directions, manual, ...

⇨ No mathematical object

What is an Algorithm?



Wikimedia, Creative Commons

230 A. M. Turing [Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number

Oxford University Press

1936.] On computable numbers. 231

have valuable applications. In particular, it is shown (§ 11) that the Hilbertian Entscheidungsproblem can have no solution.

In a recent paper Alonzo Church† has introduced an idea of "effective calculability", which is equivalent to my "computability", but is very differently defined. Church also reaches similar conclusions about the Entscheidungsproblem‡. The proof of equivalence between "computability" and "effective calculability" is outlined in an appendix to the present paper.

1. *Computing machines.*

We have said that the computable numbers are those whose decimals are calculable by finite means. This requires rather more explicit definition. No real attempt will be made to justify the definitions given until we reach § 9. For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited.

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions $q_1, q_2, \ldots, q_R$ which will be called "m-configurations". The machine is supplied with a "tape" (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the r-th, bearing the symbol $\mathfrak{S}(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its m-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the m-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair $q_n, \mathfrak{S}(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (*i.e.*

Oxford University Press

What is an Algorithm?

A mathematician has the following things available

- A pen
- An arbitrary number of sheets of checkered paper
- A finite number of rules in his head (arithmetic, logic, ...)

What is an Algorithm?

A mathematician has the following things available

- A pen
- An arbitrary number of sheets of checkered paper
- A finite number of rules in his head (arithmetic, logic, . . .)

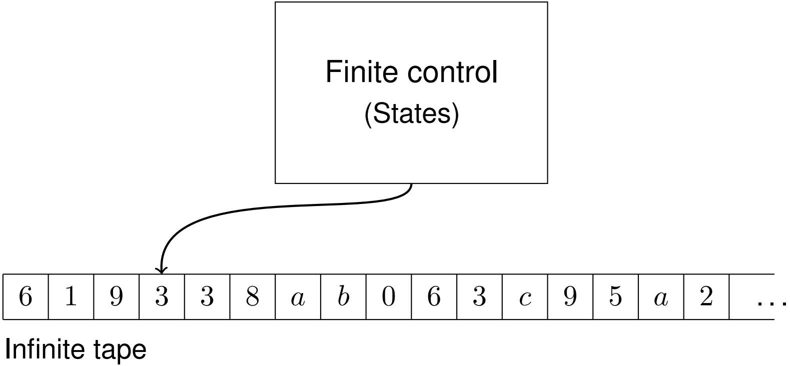
Input: String (word) written on the paper

Procedure: The mathematician. . .

- Looks a symbol (letter) to which the pen currently points
- Moves pen depending on this and writes on paper
- Eventually ends her/his work (“halts”)

Output: String (word) written on the paper

The Turing Machine



Intermission

Related Models of Computation

Related Models of Computation

Turing machine that...

- has read-only access on tape
- reads tape once from left to right

Related Models of Computation

Turing machine that...

- has read-only access on tape
- reads tape once from left to right

⇨ **Finite automaton**

- (equivalent to regular expression and regular grammars)

Related Models of Computation

Turing machine that. . .

- has read-only access on tape
- reads tape once from left to right

⇨ **Finite automaton**

- (equivalent to regular expression and regular grammars)

Turing machine that. . .

- uses a stack instead of a tape
- is allowed to make nondeterministic decisions

Related Models of Computation

Turing machine that...

- has read-only access on tape
- reads tape once from left to right

⇨ **Finite automaton**

- (equivalent to regular expression and regular grammars)

Turing machine that...

- uses a stack instead of a tape
- is allowed to make nondeterministic decisions

⇨ **Nondeterministic pushdown automaton**

- (equivalent to context-free grammars)

Wikimedia, Creative Commons

The Chomsky Hierarchy

Hierarchy of decision problems (in this context called “languages”) according to which model of computing (equivalently type of grammar) can decide them

But back to the Turing Machine. . .

Alonzo Church (1903 – 1995)

Wikimedia, Creative Commons

Church Turing Thesis

Turing machines (TMs) that halt can compute exactly what any algorithm can compute

Alonzo Church (1903 – 1995)

Wikimedia, Creative Commons

Church Turing Thesis

Turing machines (TMs) that halt can compute exactly what any algorithm can compute

The other way around

The things that TMs cannot compute can also not be computed by any algorithm (no matter whether it is written in Assembler, C, C++, Java, Python, or Ruby on Rails)

Undecidability

- TMs solve decision problems
- Is $x \in \mathrm{PRIMES}$?
- $\mathrm{PRIMES} = \{y \mid y \text{ is a prime number}\}$

Undecidability

- TMs solve decision problems
- Is $x \in \mathrm{PRIMES}$?
- $\mathrm{PRIMES} = \{y \mid y \text{ is a prime number}\}$
- Let M_{PRIMES} be a TM for PRIMES
- If $x \in \mathrm{PRIMES}$, then M_{PRIMES} "answers" YES
- If $x \notin \mathrm{PRIMES}$, then M_{PRIMES} "answers" NO

Undecidability

- TMs solve decision problems
- Is $x \in \mathrm{PRIMES}$?
- $\mathrm{PRIMES} = \{y \mid y \text{ is a prime number}\}$
- Let M_{PRIMES} be a TM for PRIMES
- If $x \in \mathrm{PRIMES}$, then M_{PRIMES} "answers" YES
- If $x \notin \mathrm{PRIMES}$, then M_{PRIMES} "answers" NO

⇨ $\mathrm{PRIMES} = \{2, 3, 5, 7, 11, 13, 17, 19, \ldots\}$

⇨ Decision problems are infinite sets of strings (words)

Undecidability

Theorem (Turing)

There are decision problems, which cannot be solved by any TM (they are therefore called **undecidable**)

Undecidability

Theorem (Turing)

There are decision problems, which cannot be solved by any TM (they are therefore called **undecidable**)

- Enumerate all TMs: $M_1, M_2, M_3, \ldots$
- Enumerate all words (over fixed alphabet): $w_1, w_2, w_3, \ldots$
- Assume every TM M_i solves some decision problem
- For every word w_j the TM M_i answers either YES or NO

Undecidability

- Again construct table
- Rows corresond to TMs, columns correspond to words

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1										
M_2										
M_3										
M_4										
$\vdots$										

Undecidability

- Again construct table
- Rows corresond to TMs, columns correspond to words

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	1	0	0	1	1	1	1	0	0	
M_2	0	0	1	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- 1 in cell (i, j) if M_i answers YES for w_j
- 0 in cell (i, j) if M_i answers NO for w_j

Undecidability

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	1	0	1	1	1	1	1	0	0	
M_2	0	0	0	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

Undecidability

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	$\boxed{1}$	0	1	1	1	1	1	0	0	
M_2	0	0	0	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 answers YES for w_1

Undecidability

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	$\boxed{1}$	0	1	1	1	1	1	0	0	
M_2	0	$\boxed{0}$	0	0	0	0	1	0	1	$\dots$
M_3	1	0	0	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 answers YES for w_1
- M_2 answers NO for w_2

Undecidability

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	$\boxed{1}$	0	1	1	1	1	1	0	0	
M_2	0	$\boxed{0}$	0	0	0	0	1	0	1	$\dots$
M_3	1	0	$\boxed{0}$	1	0	1	0	0	0	
M_4	0	0	1	1	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 answers YES for w_1
- M_2 answers NO for w_2
- M_3 answers NO for w_3

Undecidability

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	$\boxed{1}$	0	1	1	1	1	1	0	0	
M_2	0	$\boxed{0}$	0	0	0	0	1	0	1	$\dots$
M_3	1	0	$\boxed{0}$	1	0	1	0	0	0	
M_4	0	0	1	$\boxed{1}$	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 answers YES for w_1
- M_2 answers NO for w_2
- M_3 answers NO for w_3
- M_4 answers YES for w_4

Undecidability

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	$\dots$
M_1	$\boxed{1}$	0	1	1	1	1	1	0	0	
M_2	0	$\boxed{0}$	0	0	0	0	1	0	1	$\dots$
M_3	1	0	$\boxed{0}$	1	0	1	0	0	0	
M_4	0	0	1	$\boxed{1}$	0	0	1	0	0	
$\vdots$				$\vdots$						$\ddots$

- M_1 answers YES for w_1
- M_2 answers NO for w_2
- M_3 answers NO for w_3
- M_4 answers YES for w_4
- Define decision problem DIAG (**Diagonalization**)
- w_i is in DIAG if M_i answers NO for w_i
- In this example, $\mathrm{DIAG} = \{w_2, w_3, \dots\}$

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

- Suppose there is a TM M for *DIAG*

⇨ M must appear in the enumeration of all TMs

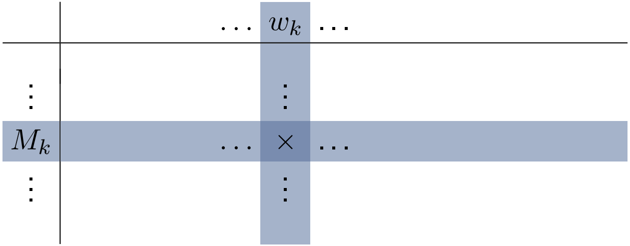
⇨ Suppose M is the kth TM, that is, M_k

The word w_i is in DIAG if and only if M_i answers NO for w_i

- Suppose there is a TM M for *DIAG*

⇨ M must appear in the enumeration of all TMs

⇨ Suppose M is the kth TM, that is, M_k; consider cell (k, k)



- Does this cell contain a 1 or a 0?

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

⇨ This means that M_k answers YES for w_k

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

- ⇨ This means that M_k answers YES for w_k
- ⇨ By construction, w_k is not DIAG

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

⇨ This means that M_k answers YES for w_k

⇨ By construction, w_k is not DIAG

- But M_k is a TM for DIAG

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

⇨ This means that M_k answers YES for w_k

⇨ By construction, w_k is not DIAG

- But M_k is a TM for DIAG

⇨ Thus, M_k answers NO for w_k

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

- ⇨ This means that M_k answers YES for w_k
- ⇨ By construction, w_k is not DIAG
- But M_k is a TM for DIAG
- ⇨ Thus, M_k answers NO for w_k

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

- ⇨ This means that M_k answers YES for w_k
- ⇨ By construction, w_k is not DIAG
- But M_k is a TM for DIAG
- ⇨ Thus, M_k answers NO for w_k

Oh well, then suppose cell (k, k) contains a 0

- ⇨ This means that M_k answers NO for w_k

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

⇨ This means that M_k answers YES for w_k

⇨ By construction, w_k is not DIAG

- But M_k is a TM for DIAG

⇨ Thus, M_k answers NO for w_k

Oh well, then suppose cell (k, k) contains a 0

⇨ This means that M_k answers NO for w_k

⇨ By construction, w_k is in DIAG

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

- ⇨ This means that M_k answers YES for w_k
- ⇨ By construction, w_k is not DIAG
  - But M_k is a TM for DIAG
- ⇨ Thus, M_k answers NO for w_k

Oh well, then suppose cell (k, k) contains a 0

- ⇨ This means that M_k answers NO for w_k
- ⇨ By construction, w_k is in DIAG
  - But M_k is a TM for DIAG

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

⇨ This means that M_k answers YES for w_k

⇨ By construction, w_k is not DIAG

- But M_k is a TM for DIAG

⇨ Thus, M_k answers NO for w_k

Oh well, then suppose cell (k, k) contains a 0

⇨ This means that M_k answers NO for w_k

⇨ By construction, w_k is in DIAG

- But M_k is a TM for DIAG

⇨ Thus, M_k answers YES on w_k

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

- ⇨ This means that M_k answers YES for w_k
- ⇨ By construction, w_k is not DIAG
  - But M_k is a TM for DIAG
- ⇨ Thus, M_k answers NO for w_k

Oh well, then suppose cell (k, k) contains a 0

- ⇨ This means that M_k answers NO for w_k
- ⇨ By construction, w_k is in DIAG
  - But M_k is a TM for DIAG
- ⇨ Thus, M_k answers YES on w_k

Undecidability

The word w_i is in DIAG if and only if M_i answers NO for w_i

Suppose cell (k, k) contains a 1

- ⇨ This means that M_k answers YES for w_k
- ⇨ By construction, w_k is not DIAG
- But M_k is a TM for DIAG
- ⇨ Thus, M_k answers NO for w_k

Oh well, then suppose cell (k, k) contains a 0

- ⇨ This means that M_k answers NO for w_k
- ⇨ By construction, w_k is in DIAG
- But M_k is a TM for DIAG
- ⇨ Thus, M_k answers YES on w_k

Contradiction

The Halting Problem

```python
print("Hello, world.")
exit()
```

The Halting Problem

```
print("Hello, world.")
exit()
```

```
while True:
    i = 1
print("Hello, world.")
exit()
```

The Halting Problem

```
print("Hello, world.")
exit()
```

```
while True:
    i = 1
print("Hello, world.")
exit()
```

```
n = 4
p = 1
while True:
    p = p + 1
    if isprime(p):
        q = n - p
        if isprime(q):
            print n + "=" + p + "+" + q
            n = n + 2
            p = 1
    if p > n/2:
        exit()
```

The Halting Problem

```
print("Hello, world.")
exit()
```

```
while True:
    i = 1
print("Hello, world.")
exit()
```

```
n = 4                                                   # Initialisation
p = 1
while True:
    p = p + 1
    if isprime(p):
        q = n - p
        if isprime(q):                                  # p and q are primes and n = p + q
            print n + "=" + p + "+" + q
            n = n + 2
            p = 1
    if p > n/2:                                         # No primes p and q with n = p + q found
        exit()
```

The Halting Problem

```
print("Hello, world.")
exit()
```

```
while True:
    i = 1
print("Hello, world.")
exit()
```

```
n = 4                                                  # Initialisation
p = 1
while True:
    p = p + 1
    if isprime(p):
        q = n - p
        if isprime(q):                    # p and q are primes and n = p + q
            print n + "=" + p + "+" + q
            n = n + 2
            p = 1
    if p > n/2:                   # No primes p and q with n = p + q found
        exit()
```

Goldbach Conjecture

Intermission

Other Things due to Turing

Other Things due to Turing

The Turing test

- Interaction (dialog) between human and computer (algorithm)
- An interrogator I talks to both of them
  (no audio visual contact)
- If I cannot tell which of them
  is the machine, computer passes test

Other Things due to Turing

The Turing test

- Interaction (dialog) between human and computer (algorithm)
- An interrogator I talks to both of them (no audio visual contact)
- If I cannot tell which of them is the machine, computer passes test

Deciphering the ENIGMA

- German encryption device used in WW2
- Turing designed the **bombe**
- German communication could be read
- Probably war-deciding

Wikimedia, Creative Commons

But again back to the Turing Machine. . .

Fast Turing Machines


Additional ABI support: linux.
Starting cron.
Local package initialization:.
Additional TCP options:.
Starting moused:.
Starting background file system checks in 60 seconds.
Fri Aug 5 18:58:36 CEST 2005
FreeBSD/i386 (unixbox) (ttyv0)
login: root
Password:
Login incorrect
SAMSUNG
1.25 Liter
Bruteforce Cat 2.0

Running Time Analysis

- Now consider decision problems that can be solved by TMs

⇨ For example, PRIMES

Running Time Analysis

- Now consider decision problems that can be solved by TMs

⇨ For example, PRIMES

- How fast can this be done?
- We are interested in the number of operations

Running Time Analysis

- Now consider decision problems that can be solved by TMs

⇨ For example, PRIMES

- How fast can this be done?
- We are interested in the number of operations
- For sure, $x \in \mathrm{PRIMES}$ can be decided faster for $x = 5$ than for $x = 10\,000\,000\,013$
- In general, the "running time" increases with the input size n
- Roughly, we have $n = \log_2 x$

Running Time Analysis

- Now consider decision problems that can be solved by TMs

⇨ For example, PRIMES

- How fast can this be done?
- We are interested in the number of operations
- For sure, $x \in \mathrm{PRIMES}$ can be decided faster for $x = 5$ than for $x = 10\,000\,000\,013$
- In general, the "running time" increases with the input size n
- Roughly, we have $n = \log_2 x$

⇨ But how fast "grows" the running time?

Running Time Analysis

n	10	50	100	300	$10\,000$
$10n$	100	500	$1\,000$	$3\,000$	$100\,000$
$4n^2$	400	$10\,000$	$40\,000$	$360\,000$	$400\,000\,000$
n^3	$1\,000$	$125\,000$	$1\,000\,000$	$27\,000\,000$	13 digits
2^n	1024	16 digits	31 digits	91 digits	$3\,011$ digits
3^n	$59\,049$	24 digits	48 digits	143 digits	$4\,772$ digits

Running Time Analysis

n	10	50	100	300	10 000
$10n$	100	500	1 000	3 000	100 000
$4n^2$	400	10 000	40 000	360 000	400 000 000
n^3	1 000	125 000	1 000 000	27 000 000	13 digits
2^n	1024	16 digits	31 digits	91 digits	3 011 digits
3^n	59 049	24 digits	48 digits	143 digits	4 772 digits

Definition

A function f is in $\mathcal{O}(g)$ for a function g if (for sufficiently large n) f only grows constantly faster than g

Running Time Analysis

n	10	50	100	300	10 000
$10n$	100	500	1 000	3 000	100 000
$4n^2$	400	10 000	40 000	360 000	400 000 000
n^3	1 000	125 000	1 000 000	27 000 000	13 digits
2^n	1024	16 digits	31 digits	91 digits	3 011 digits
3^n	59 049	24 digits	48 digits	143 digits	4 772 digits

Definition

A function f is in $\mathcal{O}(g)$ for a function g if (for sufficiently large n) f only grows constantly faster than g

- $4n^2 \in \mathcal{O}(n^2)$
- $100n^3 \in \mathcal{O}(n^3)$
- $100n^3 + 50n \in \mathcal{O}(n^3)$
- $n^3 \notin \mathcal{O}(n^2)$
- $5n \in \mathcal{O}(n^2)$
- $2^n \notin \mathcal{O}(n^2)$
- $\pi n^5 \in \mathcal{O}(n^5)$
- $n \notin \mathcal{O}(\sqrt{n})$
- $\log n \in \mathcal{O}(n)$

Running Time Analysis

Consider a simple test whether a given number is a prime number

```
test = 2
while test < num:
    res = num % test
    if res == 0:
        print("NO")
        exit()
    test += 1
print("YES")
exit()
```

Running Time Analysis

Consider a simple test whether a given number is a prime number

```
test = 2
while test < num:
    res = num % test
    if res == 0:
        print("NO")
        exit()
    test += 1
print("YES")
exit()
```

If `num` is prime, then the loop is executed roughly `num` $\approx 2^n$ times

Running Time Analysis

Consider a simple test whether a given number is a prime number

```
test = 2
while test < num:
    res = num % test
    if res == 0:
        print("NO")
        exit()
    test += 1
print("YES")
exit()
```

```
test = 2
root_num = sqrt(num)
while test <= root_num:
    res = num % test
    if res == 0:
        print("NO")
        exit()
    test += 1
print("YES")
exit()
```

If num is prime, then the loop is executed roughly num $\approx 2^n$ times

Running Time Analysis

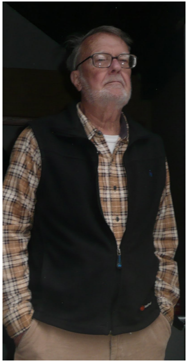
Consider a simple test whether a given number is a prime number

```
test = 2
while test < num:
    res = num % test
    if res == 0:
        print("NO")
        exit()
    test += 1
print("YES")
exit()
```

If num is prime, then the loop is executed roughly $\texttt{num} \approx 2^n$ times

```
test = 2
root_num = sqrt(num)
while test <= root_num:
    res = num % test
    if res == 0:
        print("NO")
        exit()
    test += 1
print("YES")
exit()
```

It is sufficient to test until the square root of num; loop is executed roughly $\sqrt{\texttt{num}} \approx 1.41^n$ times



Unknown

Unknown

Thesis of C. and E.

Efficient algorithms are those that run in polynomial time

This means the running time is in $\mathcal{O}(n^k)$ for some fixed $k \in \mathbb{N}$

Efficient Algorithms

- The term is independent of the concrete computing model

⇨ If a “real algorithm” can compute something in $\mathcal{O}(n^k)$, then there is a TM that can do it in $\mathcal{O}(n^{k'})$

- Therefore, we call the class of polynomials **robust**

Efficient Algorithms

- The term is independent of the concrete computing model

⇨ If a “real algorithm” can compute something in $\mathcal{O}(n^k)$, then there is a TM that can do it in $\mathcal{O}(n^{k'})$

- Therefore, we call the class of polynomials **robust**

Definition

The class $\mathcal{P}$ contains all decision problems that can be solved efficiently by TMs

Efficient Algorithms

- We are interested in the running time in the **worst case**

⇨ For our test whether a number is a prime number, this means that the input is prime

⇨ If the input is even, both of the above algorithm are fast

Efficient Algorithms

- We are interested in the running time in the **worst case**

⇨ For our test whether a number is a prime number, this means that the input is prime

⇨ If the input is even, both of the above algorithm are fast

- Our algorithms for prime numbers are both not efficient

Efficient Algorithms

- We are interested in the running time in the **worst case**

⇨ For our test whether a number is a prime number, this means that the input is prime

⇨ If the input is even, both of the above algorithm are fast

- Our algorithms for prime numbers are both not efficient
- However, an efficient algorithm exists for this decision problem

⇨ $\mathrm{PRIMES} \in \mathcal{P}$

Efficient Algorithms

- We are interested in the running time in the **worst case**

⇨ For our test whether a number is a prime number, this means that the input is prime

⇨ If the input is even, both of the above algorithm are fast

- Our algorithms for prime numbers are both not efficient
- However, an efficient algorithm exists for this decision problem

⇨ $\mathrm{PRIMES} \in \mathcal{P}$

- How about decision problems that are not in $\mathcal{P}$?

$\mathcal{NP}$-Completeness

$\mathcal{NP}$-Completeness

IND-SET

CLIQUE

Polynomial-Time Verifier

- Consider alternative model
- A **polynomial-time verifier** (PV) has to verify the solution of a problem and does not have to compute it itself

Polynomial-Time Verifier

- Consider alternative model
- A **polynomial-time verifier** (PV) has to verify the solution of a problem and does not have to compute it itself
- With the input, it gets a potential “witness”
- This witness is some string

Polynomial-Time Verifier

- Consider alternative model
- A **polynomial-time verifier** (PV) has to verify the solution of a problem and does not have to compute it itself
- With the input, it gets a potential "witness"
- This witness is some string

⇨ If the input is a "YES instance," then the witness proves this

⇨ If the input is a "NO instance," then the witness is arbitrary

Polynomial-Time Verifier

- Consider alternative model
- A **polynomial-time verifier** (PV) has to verify the solution of a problem and does not have to compute it itself
- With the input, it gets a potential "witness"
- This witness is some string

⇨ If the input is a "YES instance," then the witness proves this

⇨ If the input is a "NO instance," then the witness is arbitrary

⇨ PV has to **verify** instance with help of the witness

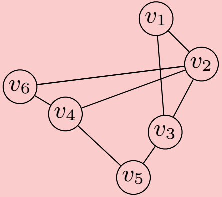
Hamiltonian Cycle HC

$\mathrm{HC} = \{y \mid y \text{ is a graph with a Hamiltonian cycle}\}$

Hamiltonian Cycle HC

$\mathrm{HC} = \{y \mid y$ is a graph with a Hamiltonian cycle$\}$

TM gets input

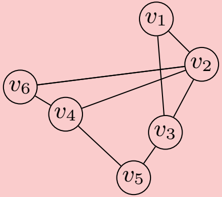


and has to compute output YES or NO

Hamiltonian Cycle HC

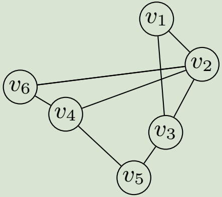
$\mathrm{HC} = \{y \mid y \text{ is a graph with a Hamiltonian cycle}\}$

TM gets input



and has to compute output YES or NO

PV gets input



and potential witness

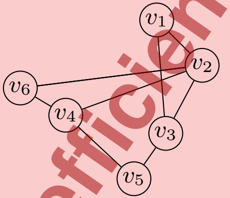
$$v_1, v_3, v_5, v_4, v_6, v_2, v_1$$

and has to verify them with YES or NO

Hamiltonian Cycle HC

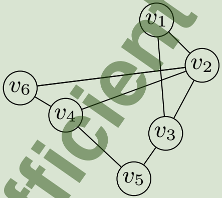
$\mathrm{HC} = \{y \mid y \text{ is a graph with a Hamiltonian cycle}\}$

TM gets input



and has to compute output YES or NO

PV gets input



and potential witness

$$v_1, v_3, v_5, v_4, v_6, v_2, v_1$$

and has to verify them with YES or NO

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

The class $\mathcal{NP}$ contains all decision problems that can be verified efficiently by PVs

Definition

The class $\mathcal{NP}$ contains all decision problems that can be verified efficiently by PVs

- "$\mathcal{N}$" stands for "nondeterministic"

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

The class $\mathcal{NP}$ contains all decision problems that can be verified efficiently by PVs

- "$\mathcal{N}$" stands for "nondeterministic"
- PVs have an obvious advantage compared to TMs
- If we cannot solve some problem efficiently with a PV, then we cannot solve it efficiently with a TM

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

The class $\mathcal{NP}$ contains all decision problems that can be verified efficiently by PVs

- "$\mathcal{N}$" stands for "nondeterministic"
- PVs have an obvious advantage compared to TMs
- If we cannot solve some problem efficiently with a PV, then we cannot solve it efficiently with a TM
- If we can solve some problem efficiently with a TM, then we can also solve it efficiently with a PV

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

The class $\mathcal{NP}$ contains all decision problems that can be verified efficiently by PVs

- "$\mathcal{N}$" stands for "nondeterministic"
- PVs have an obvious advantage compared to TMs
- If we cannot solve some problem efficiently with a PV, then we cannot solve it efficiently with a TM
- If we can solve some problem efficiently with a TM, then we can also solve it efficiently with a PV

⇨ $\mathcal{P} \subseteq \mathcal{NP}$

$\mathcal{P}$ versus $\mathcal{NP}$

Definition

The class $\mathcal{NP}$ contains all decision problems that can be verified efficiently by PVs

- "$\mathcal{N}$" stands for "nondeterministic"
- PVs have an obvious advantage compared to TMs
- If we cannot solve some problem efficiently with a PV, then we cannot solve it efficiently with a TM
- If we can solve some problem efficiently with a TM, then we can also solve it efficiently with a PV

⇨ $\mathcal{P} \subseteq \mathcal{NP}$

- But what about the opposite direction?

⇨ Until today, we do not know whether $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \subsetneq \mathcal{NP}$

$\mathcal{P}$ versus $\mathcal{NP}$

- It is commonly assumed that $\mathcal{P} \subsetneq \mathcal{NP}$

⇨ There are probably decision problems in $\mathcal{NP}$ for which there is no efficient TM

$\mathcal{P}$ versus $\mathcal{NP}$

- It is commonly assumed that $\mathcal{P} \subsetneq \mathcal{NP}$

⇨ There are probably decision problems in $\mathcal{NP}$ for which there is no efficient TM

- However, it is open for decades to find such a problem

$\mathcal{P}$ versus $\mathcal{NP}$

- It is commonly assumed that $\mathcal{P} \subsetneq \mathcal{NP}$

⇨ There are probably decision problems in $\mathcal{NP}$ for which there is no efficient TM

- However, it is open for decades to find such a problem
- So what can we do?

⇨ Find a subclass of “hardest problems” in $\mathcal{NP}$

$\mathcal{P}$ versus $\mathcal{NP}$

- It is commonly assumed that $\mathcal{P} \subsetneq \mathcal{NP}$

⇨ There are probably decision problems in $\mathcal{NP}$ for which there is no efficient TM

- However, it is open for decades to find such a problem
- So what can we do?

⇨ Find a subclass of “hardest problems” in $\mathcal{NP}$

Definition

A decision problem A in $\mathcal{NP}$ is called **$\mathcal{NP}$-complete** if the existence of an efficient TM for A implies the existence of efficient TMs for all problems in $\mathcal{NP}$

Stephen A. Cook (*1939)

Wikimedia, Creative Commons

Cook's Theorem

There is an $\mathcal{NP}$-complete decision problem

Richard M. Karp (*1935)

Wikimedia, Creative Commons

Karp's 21 Problems

There exist 21 other $\mathcal{NP}$-complete decision problems

Richard M. Karp (*1935)

Wikimedia, Creative Commons

Karp's 21 Problems

There exist 21 other $\mathcal{NP}$-complete decision problems

Today we know thousands such problems, but for none of them we can actually prove that there is no efficient TM

Polynomial-Time Reduction

Assumption

- Let A and B be two decision problems in $\mathcal{NP}$

Polynomial-Time Reduction

Assumption

- Let A and B be two decision problems in $\mathcal{NP}$
- For A, we already know that it is $\mathcal{NP}$-complete

⇨ If A can be solved efficiently, then also all other problems in $\mathcal{NP}$

Polynomial-Time Reduction

Assumption

- Let A and B be two decision problems in $\mathcal{NP}$
- For A, we already know that it is $\mathcal{NP}$-complete

⇨ If A can be solved efficiently, then also all other problems in $\mathcal{NP}$

Procedure

- Show: If B can be solved efficiently, then also A

Polynomial-Time Reduction

Assumption

- Let A and B be two decision problems in $\mathcal{NP}$
- For A, we already know that it is $\mathcal{NP}$-complete

⇨ If A can be solved efficiently, then also all other problems in $\mathcal{NP}$

Procedure

- Show: If B can be solved efficiently, then also A

⇨ If B can be solved efficiently, then also all other problems in $\mathcal{NP}$

Polynomial-Time Reduction

Assumption

- Let A and B be two decision problems in $\mathcal{NP}$
- For A, we already know that it is $\mathcal{NP}$-complete

⇨ If A can be solved efficiently, then also all other problems in $\mathcal{NP}$

Procedure

- Show: If B can be solved efficiently, then also A

⇨ If B can be solved efficiently, then also all other problems in $\mathcal{NP}$

⇨ B is $\mathcal{NP}$-complete

Polynomial-Time Reduction

Assumption

- Let A and B be two decision problems in $\mathcal{NP}$
- For A, we already know that it is $\mathcal{NP}$-complete

⇨ If A can be solved efficiently, then also all other problems in $\mathcal{NP}$

Procedure

- Show: If B can be solved efficiently, then also A

⇨ If B can be solved efficiently, then also all other problems in $\mathcal{NP}$

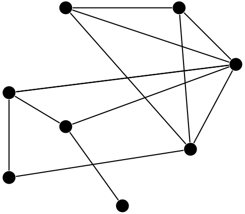
⇨ B is $\mathcal{NP}$-complete

- We "reduce" solving A efficiently to solving B efficiently

⇨ This is called a **polynomial-time reduction**

Definition

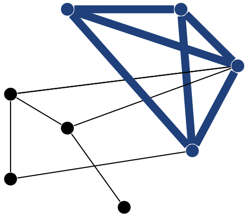
A **clique** in a graph is a subset of vertices that are all pairwise connected by edges



Polynomial-Time Reduction

Definition

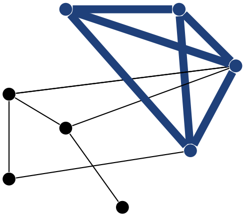
A **clique** in a graph is a subset of vertices that are all pairwise connected by edges



Polynomial-Time Reduction

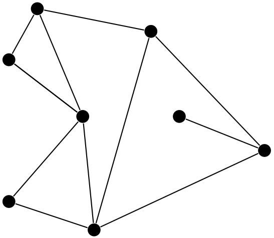
Definition

A **clique** in a graph is a subset of vertices that are all pairwise connected by edges



Definition

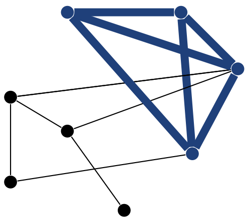
An **independent set** in a graph is a subset of vertices such that there is no edge between any of them



Polynomial-Time Reduction

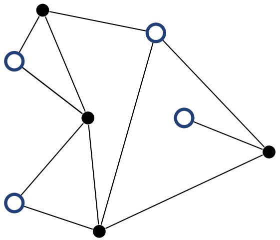
Definition

A **clique** in a graph is a subset of vertices that are all pairwise connected by edges



Definition

An **independent set** in a graph is a subset of vertices such that there is no edge between any of them



Polynomial-Time Reduction

Consider the following decision problems

- $\mathrm{CLIQUE} = \{(y,k) \mid y$ is a graph with a clique of size $k\}$
- $\mathrm{IND\text{-}SET} = \{(y,k) \mid y$ is a graph with an ind. set of size $k\}$

Polynomial-Time Reduction

Consider the following decision problems

- $\text{CLIQUE} = \{(y, k) \mid y$ is a graph with a clique of size $k\}$
- $\text{IND-SET}= \{(y, k) \mid y$ is a graph with an ind. set of size $k\}$

Procedure

- Suppose we know that CLIQUE is $\mathcal{NP}$-complete

⇨ Show that then IND-SET is also $\mathcal{NP}$-complete

Polynomial-Time Reduction

Consider the following decision problems

- $\mathrm{CLIQUE} = \{(y,k) \mid y$ is a graph with a clique of size $k\}$
- $\mathrm{IND\text{-}SET} = \{(y,k) \mid y$ is a graph with an ind. set of size $k\}$

Procedure

- Suppose we know that CLIQUE is $\mathcal{NP}$-complete

⇨ Show that then $\mathrm{IND\text{-}SET}$ is also $\mathcal{NP}$-complete

⇨ In other words we "reduce CLIQUE to $\mathrm{IND\text{-}SET}$"

Polynomial-Time Reduction

Consider the following decision problems

- $\mathrm{CLIQUE} = \{(y,k) \mid y$ is a graph with a clique of size $k\}$
- $\mathrm{IND\text{-}SET} = \{(y,k) \mid y$ is a graph with an ind. set of size $k\}$

Procedure

- Suppose we know that CLIQUE is $\mathcal{NP}$-complete

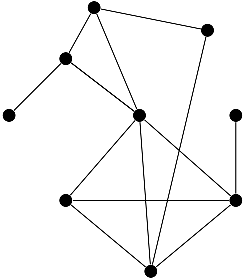
⇨ Show that then $\mathrm{IND\text{-}SET}$ is also $\mathcal{NP}$-complete

⇨ In other words we "reduce CLIQUE to $\mathrm{IND\text{-}SET}$"

⇨ If we could solve $\mathrm{IND\text{-}SET}$ efficiently, then also CLIQUE

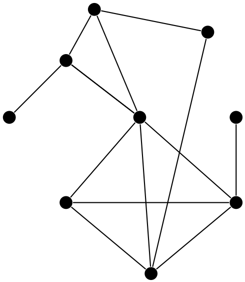
Polynomial-Time Reduction

We make the following observation



Polynomial-Time Reduction

We make the following observation

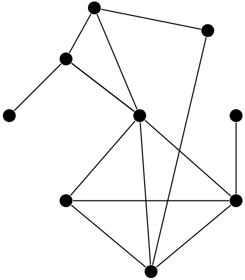


Complement graph

(that is, delete edges and
add edges where
there weren't any)

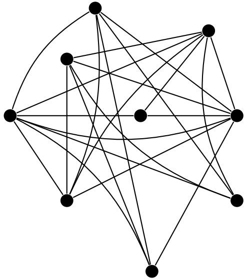
Polynomial-Time Reduction

We make the following observation



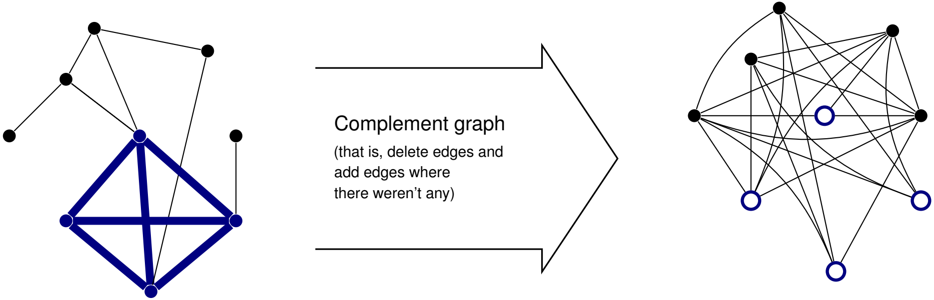
Complement graph

(that is, delete edges and
add edges where
there weren't any)



Polynomial-Time Reduction

We make the following observation



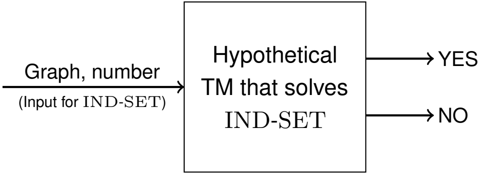
Clique of size k becomes independent set of size k

Polynomial-Time Reduction

Hypothetical
TM that solves
$\mathrm{IND\text{-}SET}$

- Suppose we have an efficient TM for $\mathrm{IND\text{-}SET}$

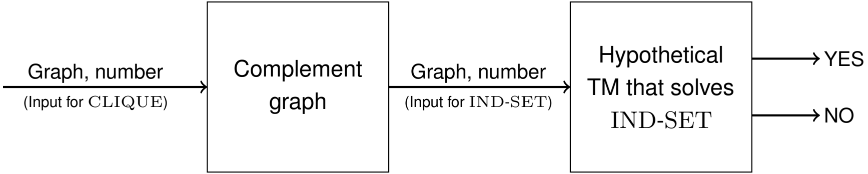
Polynomial-Time Reduction



- Suppose we have an efficient TM for IND-SET

⇨ Gets input and outputs YES or NO

Polynomial-Time Reduction

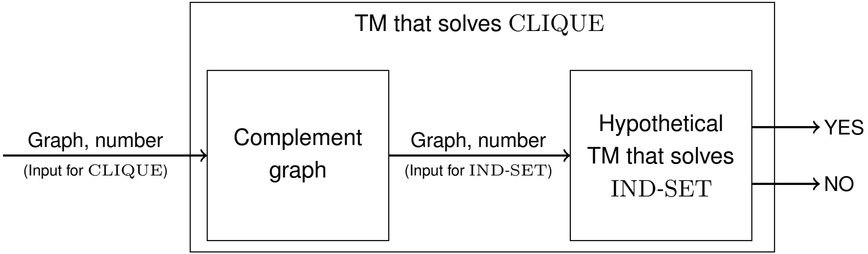


- Suppose we have an efficient TM for IND-SET

⇨ Gets input and outputs YES or NO

- Use program for complementing graph beforehand

Polynomial-Time Reduction



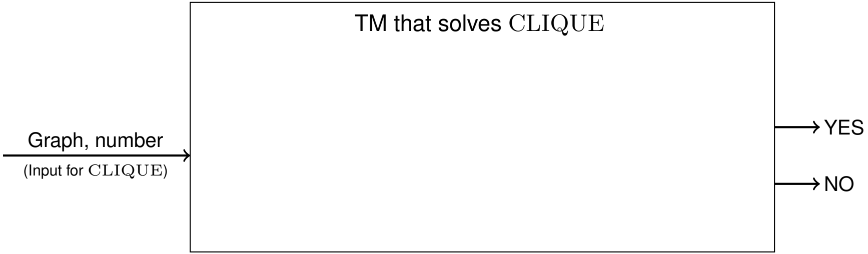
- Suppose we have an efficient TM for IND-SET

⇨ Gets input and outputs YES or NO

- Use program for complementing graph beforehand

⇨ We get an efficient TM for CLIQUE

Polynomial-Time Reduction



- Suppose we have an efficient TM for IND-SET

⇨ Gets input and outputs YES or NO

- Use program for complementing graph beforehand

⇨ We get an efficient TM for CLIQUE

Polynomial-Time Reduction

- This way, thousands of $\mathcal{NP}$-complete problems could be identified

Polynomial-Time Reduction

- This way, thousands of $\mathcal{NP}$-complete problems could be identified
- If, at some point, an efficient TM for one of them is found, then there are efficient TMs for all of them

Polynomial-Time Reduction

- This way, thousands of $\mathcal{NP}$-complete problems could be identified
- If, at some point, an efficient TM for one of them is found, then there are efficient TMs for all of them

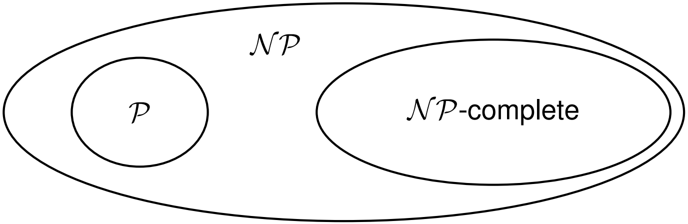
⇨ $\mathcal{P} = \mathcal{NP}$

Polynomial-Time Reduction

- This way, thousands of $\mathcal{NP}$-complete problems could be identified
- If, at some point, an efficient TM for one of them is found, then there are efficient TMs for all of them

⇨ $\mathcal{P} = \mathcal{NP}$

- Today, we assume the following relation



To sum up…

To sum up...

- There are problems that cannot be solved algorithmically

To sum up...

- There are problems that cannot be solved algorithmically
- Results on decision problems can easily be extended to “search problems”

To sum up...

- There are problems that cannot be solved algorithmically
- Results on decision problems can easily be extended to “search problems”
- The problems that can be solved can be solved differently well

To sum up…

- There are problems that cannot be solved algorithmically
- Results on decision problems can easily be extended to “search problems”
- The problems that can be solved can be solved differently well
- “Efficient” means “in polynomial time”

To sum up...

- There are problems that cannot be solved algorithmically
- Results on decision problems can easily be extended to “search problems”
- The problems that can be solved can be solved differently well
- “Efficient” means “in polynomial time”
- For many problems, we do not know any efficient algorithms

To sum up...

- There are problems that cannot be solved algorithmically
- Results on decision problems can easily be extended to “search problems”
- The problems that can be solved can be solved differently well
- “Efficient” means “in polynomial time”
- For many problems, we do not know any efficient algorithms

$\Rightarrow$ $\mathrm{HC}, \mathrm{CLIQUE}, \mathrm{IND\text{-}SET}, \ldots$

To sum up...

- There are problems that cannot be solved algorithmically
- Results on decision problems can easily be extended to "search problems"
- The problems that can be solved can be solved differently well
- "Efficient" means "in polynomial time"
- For many problems, we do not know any efficient algorithms

⇨ $\mathrm{HC}, \mathrm{CLIQUE}, \mathrm{IND\text{-}SET}, \ldots$

- However, we so far failed to prove that such algorithms do really not exist

To sum up. . .

- There are problems that cannot be solved algorithmically
- Results on decision problems can easily be extended to “search problems”
- The problems that can be solved can be solved differently well
- “Efficient” means “in polynomial time”
- For many problems, we do not know any efficient algorithms

⇨ $\mathrm{HC}, \mathrm{CLIQUE}, \mathrm{IND\text{-}SET}, \ldots$

- However, we so far failed to prove that such algorithms do really not exist
- $\mathcal{P}$ versus $\mathcal{NP}$ problem open for decades

To sum up...

- There are problems that cannot be solved algorithmically
- Results on decision problems can easily be extended to “search problems”
- The problems that can be solved can be solved differently well
- “Efficient” means “in polynomial time”
- For many problems, we do not know any efficient algorithms

⇨ $\mathrm{HC}, \mathrm{CLIQUE}, \mathrm{IND\text{-}SET}, \ldots$

- However, we so far failed to prove that such algorithms do really not exist
- $\mathcal{P}$ versus $\mathcal{NP}$ problem open for decades
- One of seven “millennium problems” ($1\,000\,000$ USD reward)

Thanks for the Attention