Computability: Three Millenia and Counting

Jon Kleinberg

Modeling Computation

Goal: Model the notion of computation mathematically.

- Analogy to physics: if we build a mathematical model, we can derive laws that guide the practice of computer science.

Five stops on the path to a theory of computation:

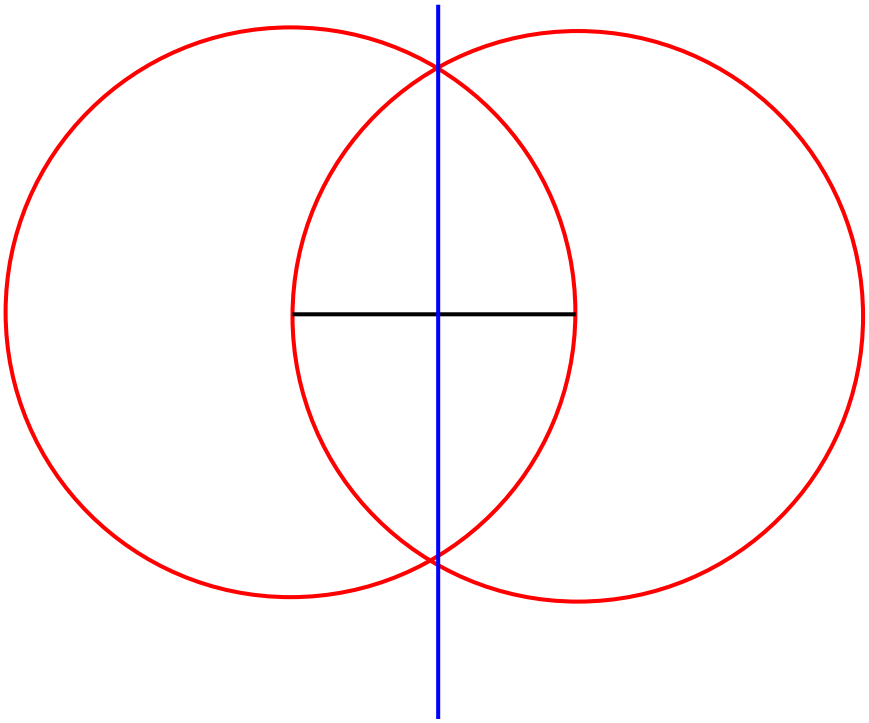
- Ancient Greece.
- Europe, early 19th century.
- 1930's.
- 1960's.
- Today.

1. Ancient Greece

Puzzle: I give you a table with a bunch of artist's implements, but no way to measure things accurately.

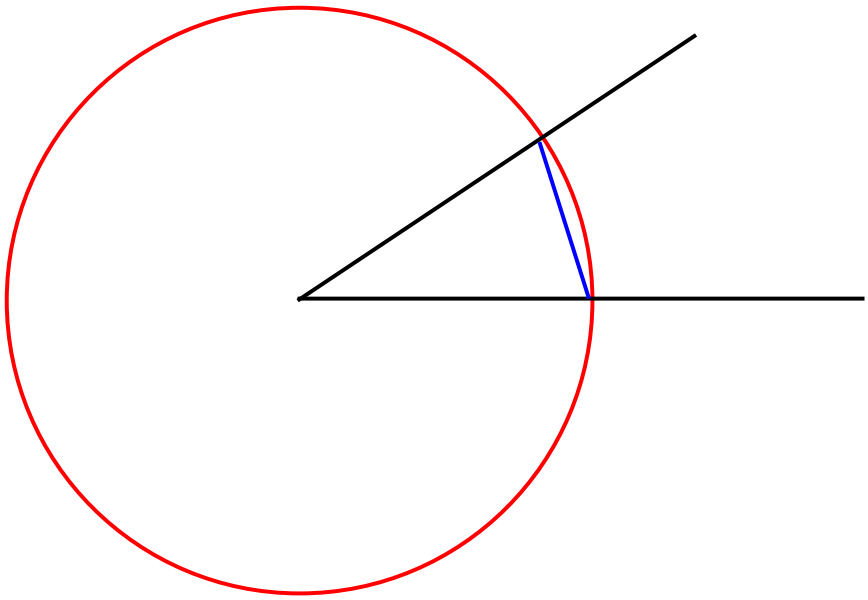
Can you put a mark on the exact midpoint of the line segment below?

A Bisection Method



- Draw a circle centered at each endpoint.
- Connect the two meeting points with a blue segment.
- Theorem: The desired point is where the blue segment meets the black segment.

Bisecting an Angle



- Draw a circle at the hinge point of the angle, mark where it meets the two arms.
- Connect the two meeting points with a blue segment.
- Theorem: the ray from the hinge through the midpoint of the blue segment bisects the original angle.

Principle #1:

"Reducing" a problem to one that you've already solved.

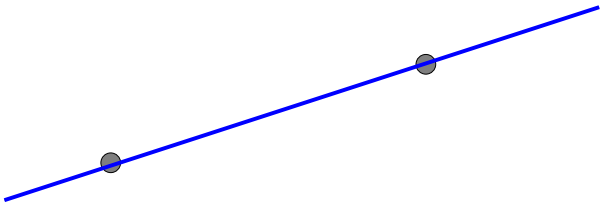
How about trisecting an angle? . . .

A General Framework

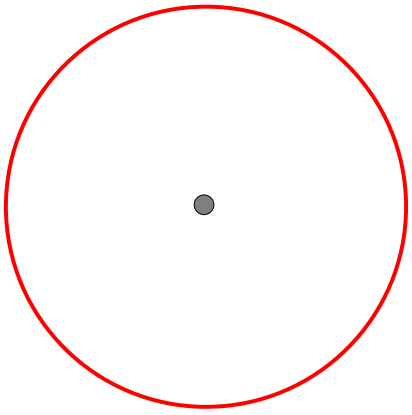
People built up a large arsenal of these constructions.

Capturing the class of “valid” constructions:

Rule 1:



Rule 2:



Principle #2:

Precisely modeling a stylized form of computation.

In a Different Vein

Another interest of Greek mathematics:

Logic and philosophy.

Modeling the truth and falsehood of statements.

This subject can be hard on one's intution:

- Epimenides, from the island of Crete, announces:

  "All Cretans are liars."

- Is this statement true or false?

  ▷ Can it be true? Then Epimenides himself would have to be a liar, in which case it would be false — a contradiction.

  ▷ So it must be false. No problem, except that it means there must be a truth-telling Cretan out there somewhere.

Truth and Falsehood

The utterance of a false sentence by someone from Crete implies that there must exist a truth-telling Cretan.

This line of reasoning seems to be leading to trouble ...

Consider the following statement:

- “I am lying.”

Is this statement true or false?

Principle #3:

The notion of self-reference:

a dangerously powerful construct.

2. Europe, early 1800's

The development of abstract algebra draws much motivation from classical geometry.

An application of abstract algebra:

- Theorem: There does not exist a sequence of compass-and-straight-edge operations that starts with an arbitrary angle and correctly produces a trisection of it.

Principle #4:

Proving that a problem is "uncomputable"
in a precisely formulated model of computation.

Must make the model precise before such a result is possible.

An Impossibility Result

- This does not say that no angle can be trisected by compass-and-straight-edge operations.
  - ▷ For example, it is not difficult to trisect a 90-degree angle.
- Rather, it says that no sequence of operations works for every angle.
- Equivalently:
  - ▷ For every sequence of compass-and-straight-edge operations, there exists an angle on which it does not produce the correct trisection.

Another Use of Algebra

Fact (from high school):

- Every quadratic equation of the form $ax^2 + bx + c = 0$ has the two roots

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

This was discovered independently in many cultures.

Much later:

- Similar formulas for general cubic (degree-3) and quartic (degree-4) equations were discovered $\approx 400 - 500$ years ago.

- What about a formula for quintic (degree-5) equations?

Quintic Formula

Principle #2 $\longrightarrow$

- We should make precise what we mean by a "formula" for one of these equations.

Definition: A quintic formula is a finite sequence of steps that produces the roots of

$$ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0$$

starting from the six coefficients. Each step can be either an arithmetic operation or the extraction of an n^{th} root.

Now, Principle #4:

- Theorem [Abel, Galois]: There is no quintic formula.

General Computation

These models of computation are highly stylized.

Also: Missing some crucial components:

- They just model straight-line "recipes."
- What about loops?

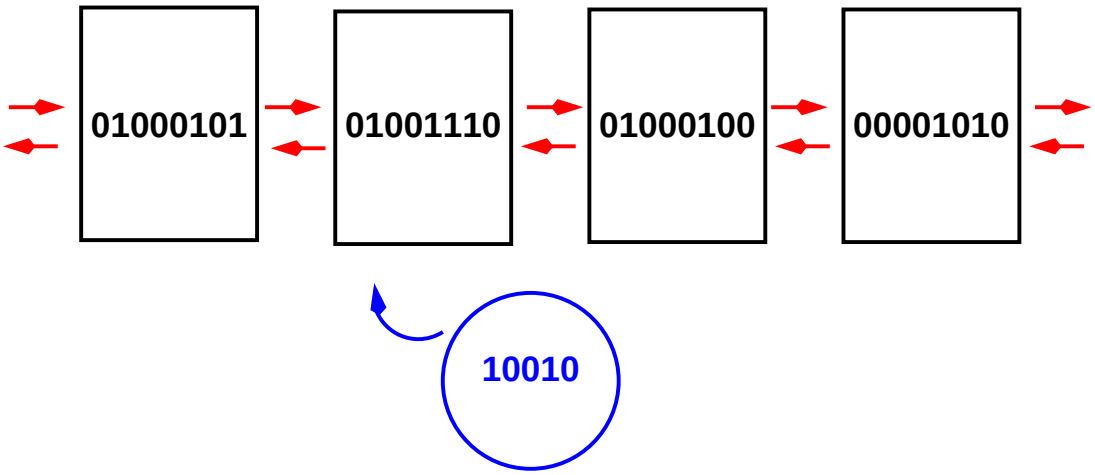
  Repetitive behavior:

  "Do X until Y happens."
- What about conditionals?

  Adaptive behavior:

  "If Y happens then do X; else do Z."

3. The Turing Machine

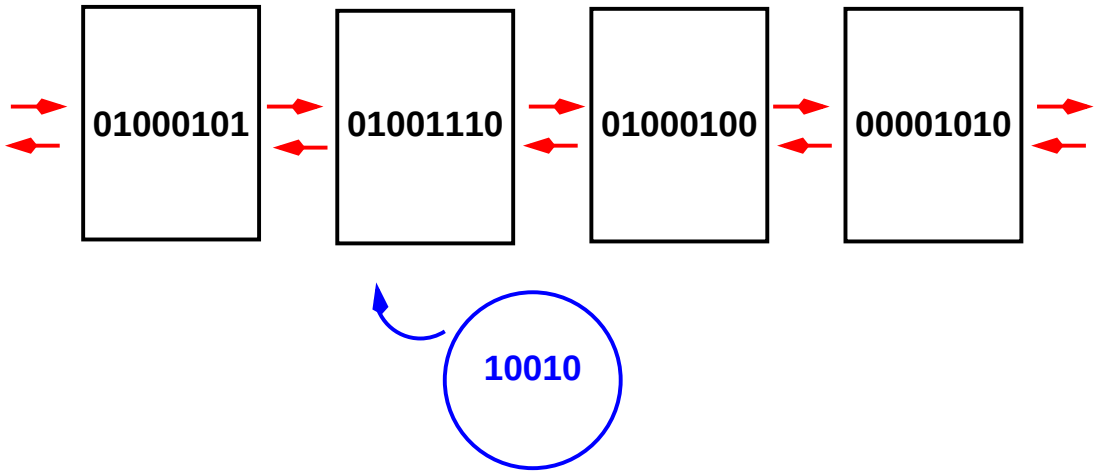


1936: Alan Turing says, “Let’s model general computation.”

Motivating image: person performing calculations on paper.

- Infinite supply of ordered pages, initialized with input.
  Each can hold a finite number of symbols.
- A control unit, reading one of the pages.
  Can be in one of a finite number of “states.”
  Based on its state and what it reads,
  it can write something and move.
- Eventually, the machine may produce output and stop.

The Turing Machine



Generality:

- First of all, it follows common sense.
- Can assume very small number of bits per page (e.g. 3) and very small number of states (e.g. 5).
- Doesn't increase model's power to allow pages to be arranged in a "web" rather than a line.
- Equivalent to:

  C and other modern programming languages

  low-level assembly code

  lambda-calculus, term-rewriting

  ...

A Universal Program

Theorem [Turing]: There is a universal program P^* for Turing machines that does the following:

- As input, it gets a description of an arbitrary program P and an input string n to P.
- As output, it writes down the output of P on n (if any).

We'd call this an "interpreter" for a language.

It's easy to miss two amazing points:

- Principle #5:

  Programs and data are really the same thing.

- Principle #6:

  The idea of a universal program, which takes a description of any other program and runs it.

Programs, Data, and Universality

- Mobile code like java applets:

  What's the code and what's the data?

- Windows e-mail viruses:

  Why is the machine in my office completely safe?

- The World Wide Web:

  How did it become a new medium in just 6 years?

  (It took radio, TV, and the phone much longer.)

Biological Systems

Biological systems implemented Principles 5 and 6 long ago.

- RNA can be read like data,
  but also used for biological structure and function.
- The ribosome: a universal machine!
  A complex of RNA and proteins that
  reads RNA and produces proteins.

A living cell is a chemical system in which the information content is explicit.

Limitations of Universal Programs

Two programs that decide if input n is a perfect square:

- Program P:
  - ▷ Try all natural numbers $x = 0, 1, 2, 3, \ldots$ in sequence.
  - ▷ If you ever reach x so that $x * x = n$, report "yes."
- Program P'
  - ▷ Try all natural numbers x from 0 to n.
  - ▷ If $x * x = n$ for any of them, report "yes."
  - ▷ Else report "no."

The point: On inputs that cause P to run forever, the universal program P^* simulating Program 1 also runs forever.

It doesn't "short-cut" the execution, or automatically convert it to P'.

Limitations of Universal Programs

We'd like a better kind of universal program — call it P^{++}.

- Given a description of a program P, and an input n, it shouldn't directly simulate P on n.
- Instead, it should run for a finite number of steps and then report the result of running P on n.
  — Denote this result $P(n)$.
- The result $P(n)$ is:
  - ▷ The output of P on n, assuming P actually stops and produces output; or
  - ▷ A special symbol to denote that P runs forever on n.

We might call this a "universal program analyzer."

An Impossibility Result

Theorem [Turing]: There is no universal program analyzer.

Why? Self-reference . . .

A universal analyzer would be a program:

It could run on itself as input, see what it does,

and then invert its answer.

Concretely:

- The set of all programs can be listed $P_1, P_2, P_3, \ldots$
- Suppose there were a universal program analyzer P^{++}.
- Consider the following program Q:
  - ▷ On input n, it determines the result of P_n running on n.
  - ▷ If $P_n(n)$ is an output value v, Q outputs $v + 1$.
  - ▷ If P_n runs forever on n, Q outputs 0.

An Impossibility Result

	1	2	3	4	...
P_1	~~3~~ 4	forever	0	0	
P_2	1	~~forever~~ 0	2	2	
P_3	1	1	~~1~~ 2	forever	
P_4	3	forever	0	~~0~~ 1	
...					

Why the program Q leads to trouble:

- Since Q is a program, it is on our list:
  $Q = P_i$ for some i.
- But this is not possible, since Q is explicitly designed so that the result $Q(i)$ is different from the result $P_i(i)$.
- Thus, no such Q can exist,
  and so no such analyzer P^{++} can exist.

A “diagonalization” technique [Cantor]

Consequences of Impossibility

Consider the following types of programs.

- A universal verifier:
  Given a program P, and a "bad" output value v, determine whether there's any input n that causes P to output v.
- A universal equivalence tester:
  Given two programs P and P', decide whether for every input, they have the same result.

We can show that given any of these programs, we'd be able to build a universal analyzer.

So none of these problems is computationally solvable.

(The Reduction Principle.)

4. Computational Complexity

Beyond the basic question of computability:

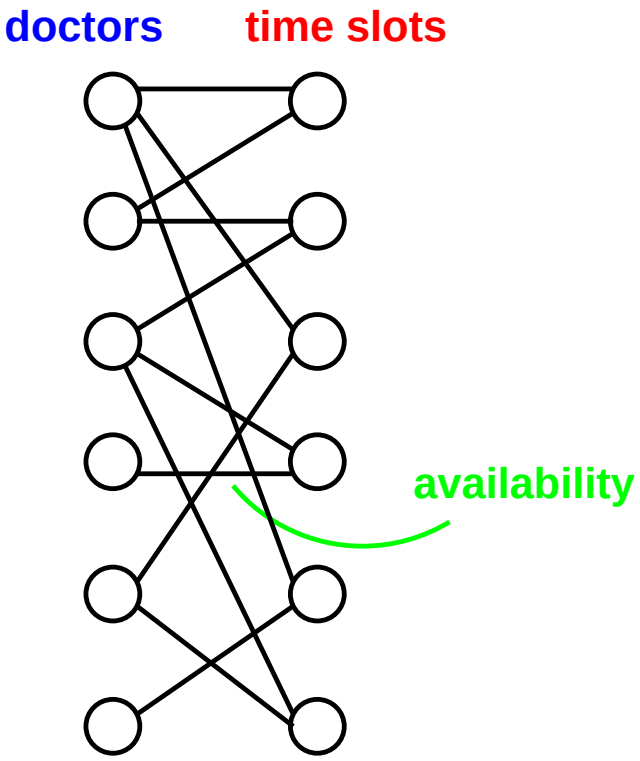
How quickly can a problem be solved?

How does its solution scale with input size?

Edmonds, Hartmanis, Stearns, . . . [1960's]:

A systematic study of computational complexity.

Some Computational Problems

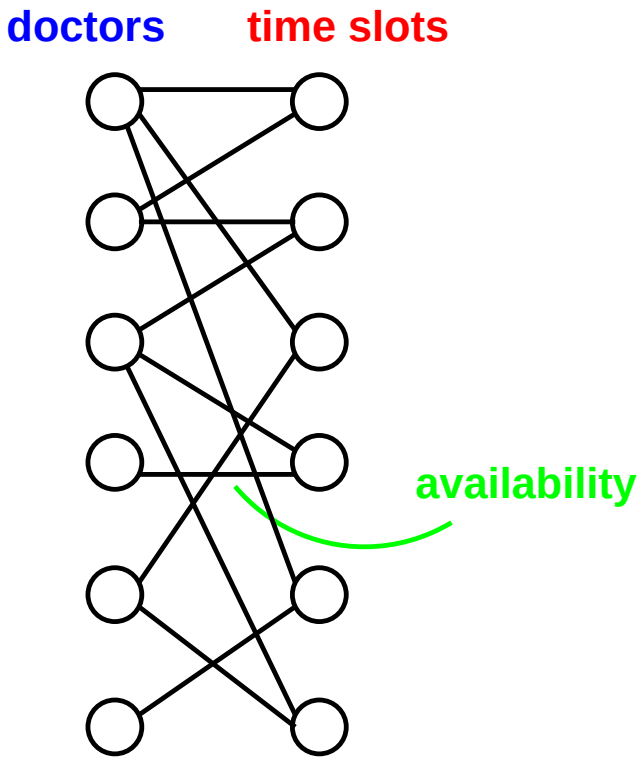


Two different problems.

- A Matching Problem: Assign each doctor to a distinct time slot when he or she is available.
- A Covering Problem: Choose a team of as few doctors as possible, so that every time slot is covered by someone on the team.

Both problems have a “search space” that is a huge, exponential function of the number of objects.

A Contrast in Complexity

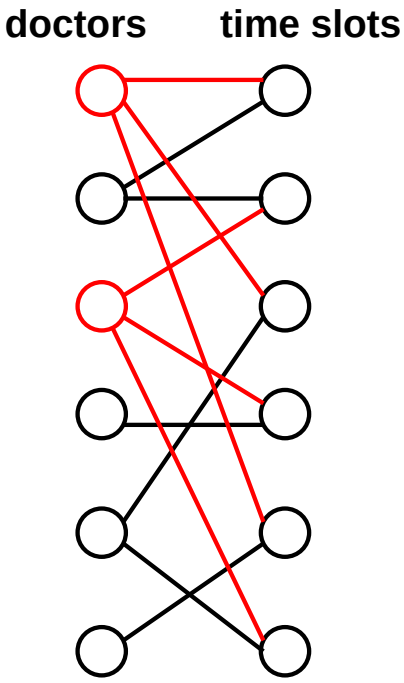


- There is a very efficient algorithm for Matching.
  (Can solve instances of size 10,000 on a PC.)
- No efficient algorithm is known for the Covering problem, and there is evidence that none exists.
  (An instance of size 100 may be much too large.)

Principle #7:

Computational complexity is more than just having a large space of possible solutions.

Some Computational Problems



But solutions to Covering can be checked.

- P: problems solvable in time polynomial in input size.
- NP: class of problems for which solutions can be checked in time polynomial in input size.
- Major open question: Is $P \neq NP$?

Principle #8:

The distinction between solving a problem and verifying a proposed solution.

5. The Present

Complexity is a major theme in computer science today:

- We do not fully understand what makes a problem hard to compute.
- Applications in optimization and search:
  building a reliable network, managing an airline, artificial intelligence, data mining . . .
- Applications in cryptography:
  protecting secrets by making them computationally hard to decrypt.