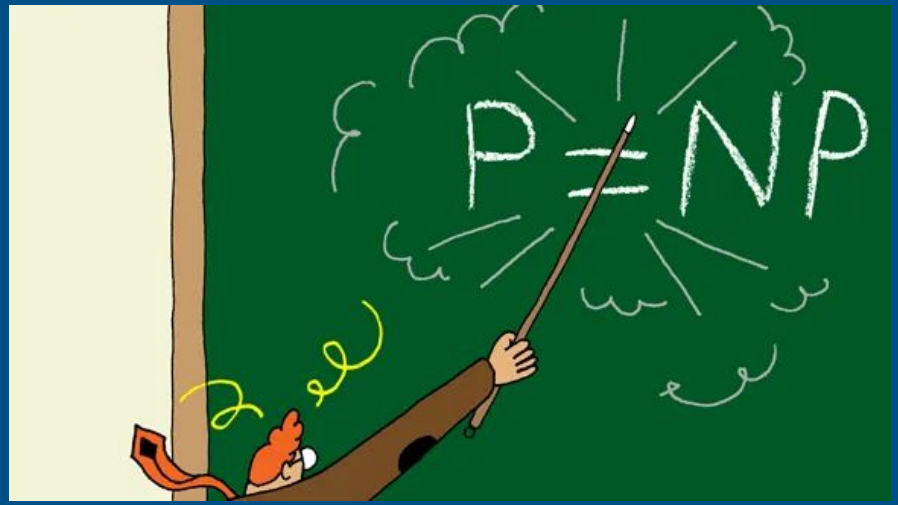
07/01/2026

P vs NP

How to become a millionaire

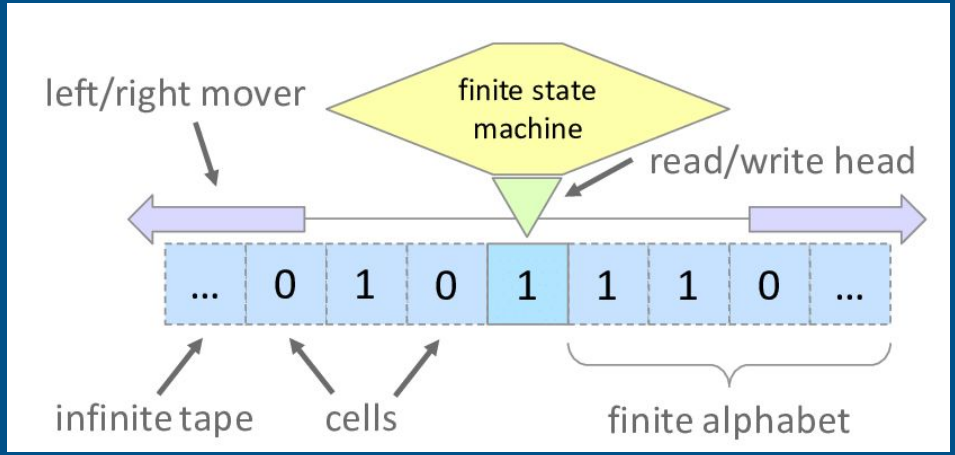


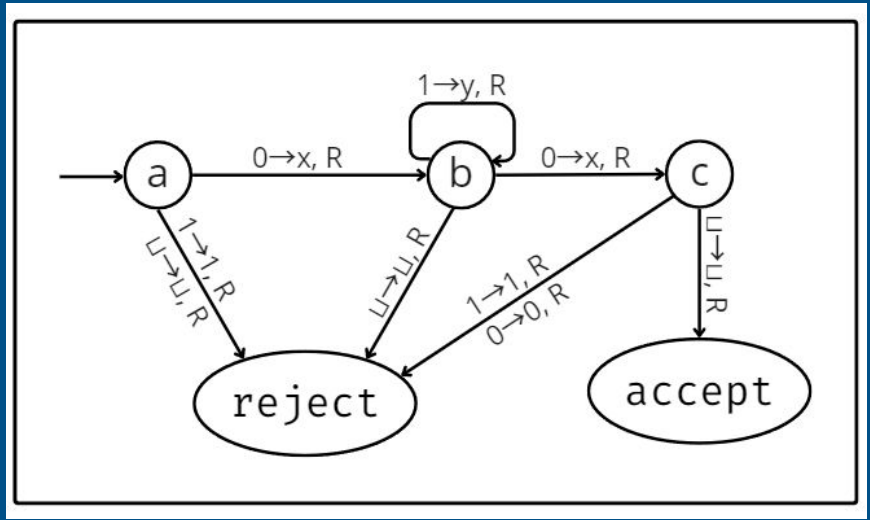
Blair McCormick

1. Turing Machines
2. What are P and NP?
3. NP-complete & NP-hard
4. Example Problems
5. Other Complexity Classes

Turing Machines

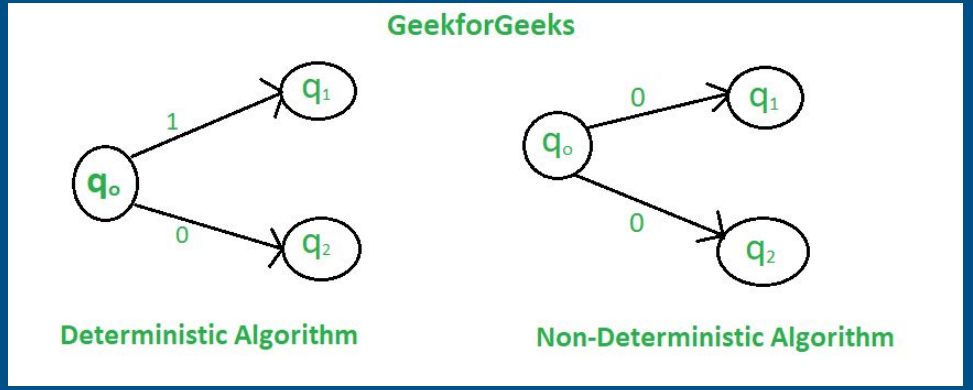
- Turing Machines (TMs) are the fundamental model of computing developed by Alan Turing in 1936.
- TMs consist of an infinite tape over some alphabet **Γ**, a set of states **Q** and a transition function **δ** that decides which state to move to based on the current state and the input on the tape.
- TMs decide a decision problem, they either accept or reject an input. The set of inputs a TM accepts is called a language. In the example this language is 01*0 or {00, 010, 0110...}.
- Very broadly, Turing proved that all “decidable” languages can be decided using some deterministic TM.Note that he found many languages that are “undecidable”, the halting problem is a famous example.
- Computation models that can simulate any TM are called Universal Turing Machines or Turing Complete. Programming languages are Turing Complete but so are things like Power Point, Minecraft and Conway’s game of life.

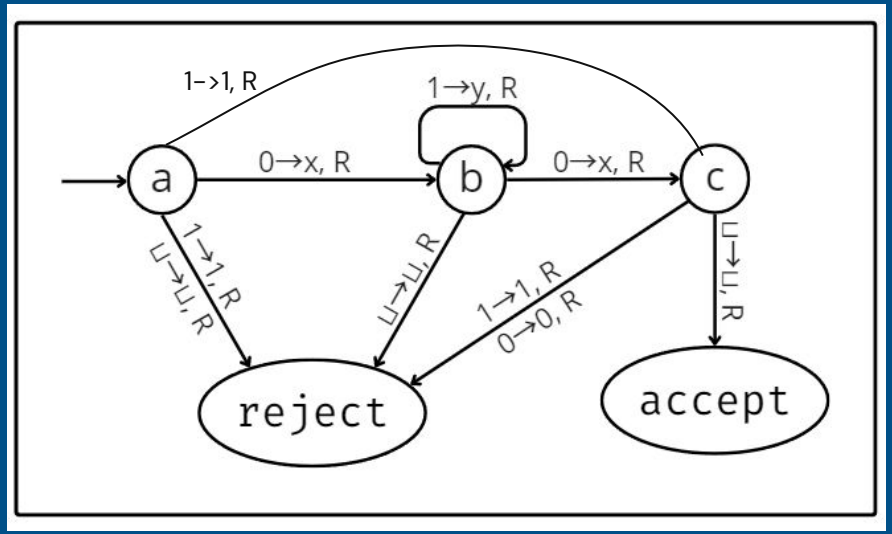




Non-Deterministic Turing Machines

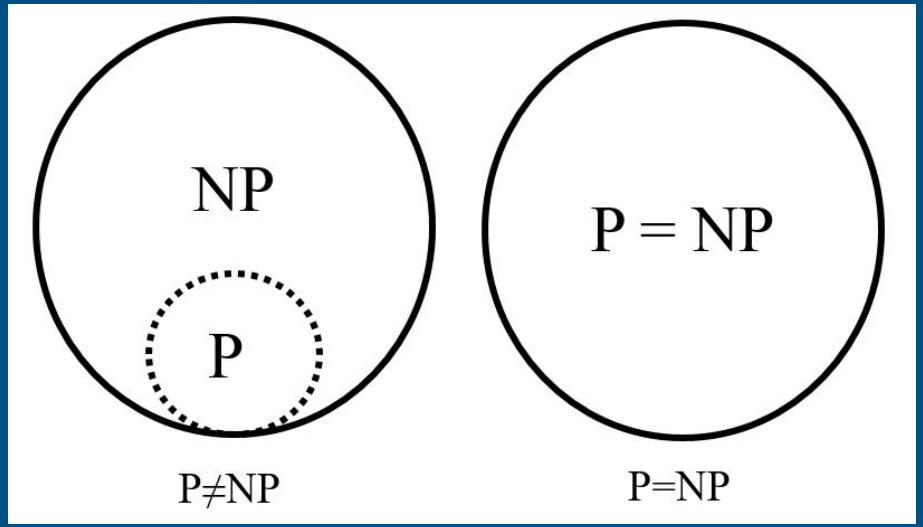
- The turing machines we have seen are deterministic; given some input we will always get the same output.
- TMs can be non-deterministic. We say that a non-deterministic turing machine accepts if there is at least one possible path of transitions that accepts.





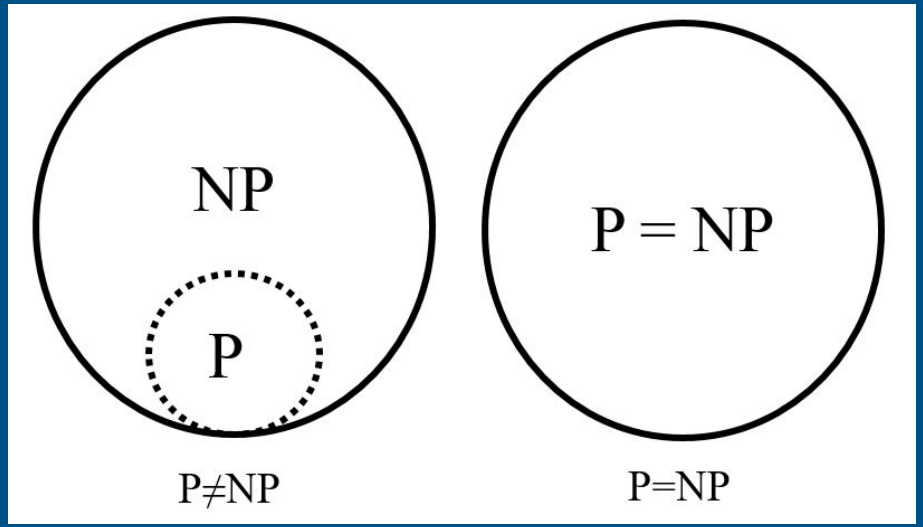
So what are P and NP?

- P and NP are Complexity Classes. Informally, this is a set of computational problems with similar complexity.
- For example, **P** is the class of tractable problems. That is they can be solved in Polynomial time.
- Similar to Big-O notation there is no distinction between $O(n^2)$ and $O(n^{200})$ problems. Both of these belong to P.
- More concretely, P is the set of all Languages solvable by a deterministic Turing Machine in a polynomial number of transitions.
- NP is the class of problems where a yes-instance can be verified in Polynomial Time given some witness. For example, consider the Travelling Salesman decision problem, it is hard to find routes but easy to verify them.
- More formally, NP is the set of Languages solvable by a non-deterministic Turing Machine in a polynomial number of transitions.
- First Theorem: P ⊆ NP. The hard bit is the other way.



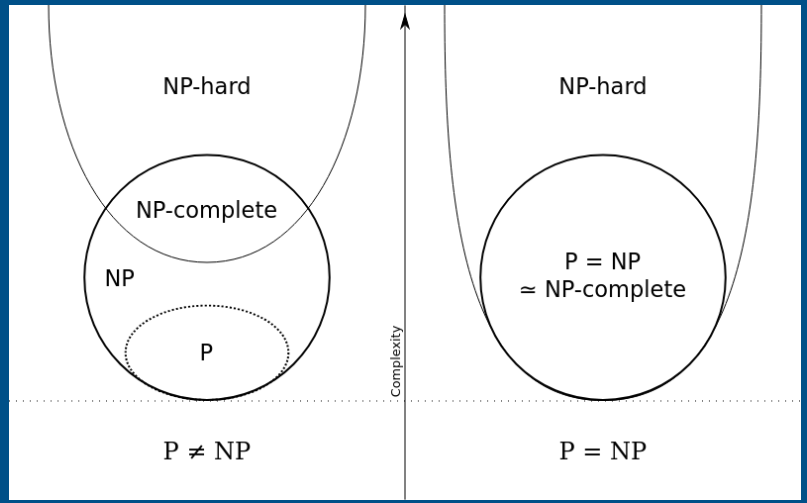
Million Dollar Question

- The millennium prize problem asks for a proof that P=NP or otherwise.
- Intuitively, this question is asking how much non-determinism helps. Ie What problems can non-deterministic algorithms solve that deterministic algorithm can't (in polynomial time)?
- What is your intuition? Does P=NP?



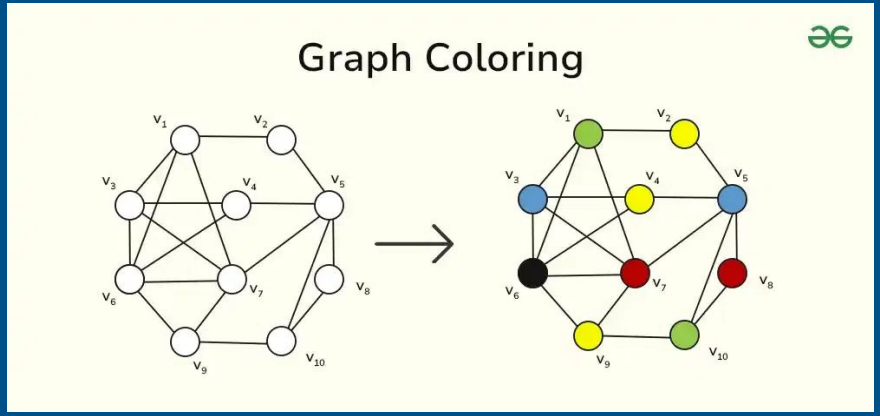
NP-Complete & NP-Hard

- **NP-Complete** problems are the hardest problems in NP. That is every problem in NP can be “reduced” to these problems in polynomial time.
- A “reduction” is just an algorithm that transforms the original input into a new input for a different problem. For example, converting some graph problem into a DFS/BFS problem is a reduction.
- **NP-Hard** problems are the same but need not be in NP. They are at least as hard as problems in **NP-Complete**
- Having Upper Bounds is really helpful for proving P=NP or otherwise. For example, if we can find a polynomial algorithm for any NP-complete or NP-hard problem then, using the polynomial “reductions”, we have polynomial algorithms for every problem in NP and thus P=NP.
- This means that NP-Complete problems are the low-hanging fruit. These are the easiest hard problems that we can solve in polynomial time to show P=NP.



Example NP-Complete Problem: 3-Colouring

- Problem Statement: Given a graph **G** does a valid 3-colouring exist?
- A valid 3-colouring assigns each node in **G** a colour such that no connected nodes have the same colour.
- 2-colouring is in **P**, can you think of a proof?



Example NP-Complete Problem: 3-Satisfiability

- Problem Statement: Given a boolean expression is there some configuration of input variables that satisfy the expression?
- Expression is in CNF form, as a conjunction of clauses which are disjunction of at most 3 literals. There is no limit on the number of variables or clauses.
- Can you see how these problems might be reduced to one another?

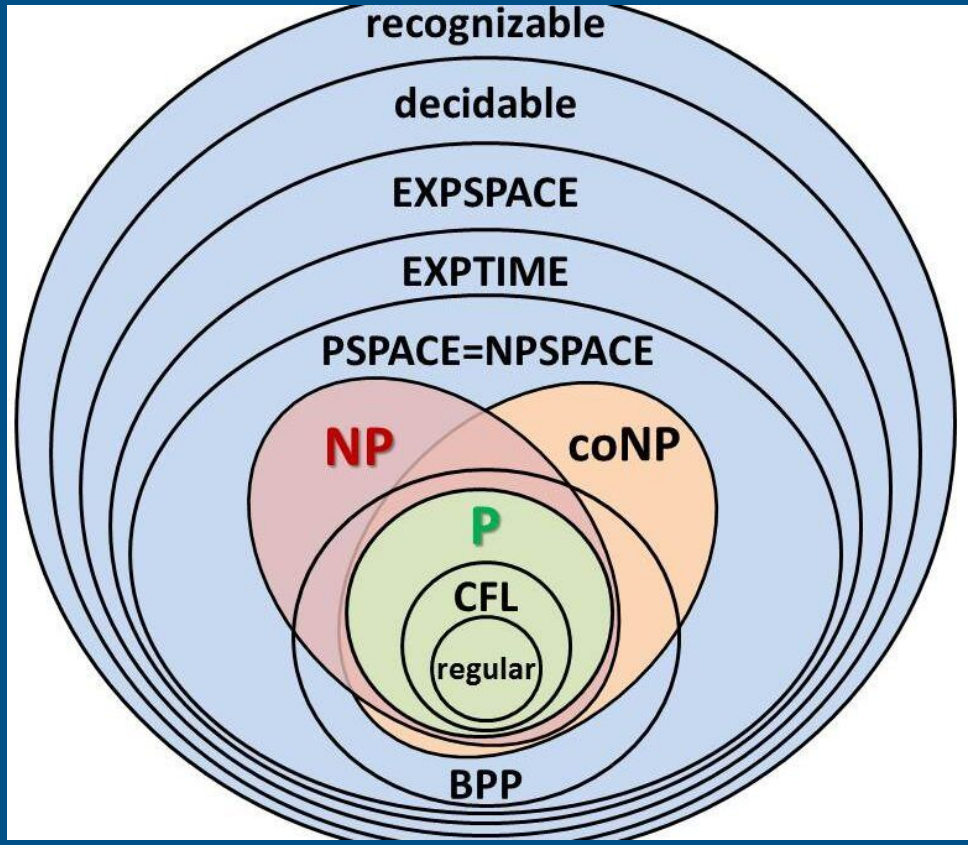
$$(x \text{ OR } y \text{ OR } z) \text{ AND } (x \text{ OR } \bar{y} \text{ OR } z) \text{ AND}$$

$$(x \text{ OR } y \text{ OR } \bar{z}) \text{ AND } (x \text{ OR } \bar{y} \text{ OR } \bar{z}) \text{ AND}$$

$$(\bar{x} \text{ OR } y \text{ OR } z) \text{ AND } (\bar{x} \text{ OR } \bar{y} \text{ OR } \bar{z})$$

Other Complexity Classes

- There are almost 500 different complexity classes so it can get quite confusing.
- **coNP:** The complement class to NP, instead of verifying yes-instances, can we verify no-instances in poly-time.
- **EXPTIME:** Solvable in O (2^n^k), absolutely massive.
- **PSPACE:** Instead of bounding time, bound space to be at most polynomial with no limit on time. Savitch's theorem showed that PSPACE=NPSPACE, ie. non-determinism doesn't help space.
- **BPP:** Solvable polynomial time by a randomised algorithm with small error.
- **NC:** Short for Nick's Class, this is the class of problems solvable in polylogarithmic time on a parallel computer with polynomial number of processors. NC ⊆ P.
- The Complexity Zoo is a fantastic resource indexing all the different classes and related theorems. If you have found this talk interesting definitely check out the introductory essay and the petting zoo.



Thank you!

Questions?