Decision problems

A decision problem is a yes or no question that you can ask a computer to solve.

Some examples:

- Does a given graph have an *Eulerian circuit*?
- Does a given graph have a *Hamiltonian cycle*?
- Is a given number *prime*?
- Given a partially filled Sudoku board, is there a *legal solution*?

Some decision problems can be solved quickly (Eulerian circuit, prime number), while others can take an exponentially long time to solve (Hamiltonian cycle).
Hamiltonian cycles

Suppose we are given a graph on \( n \) vertices and want to see if it has a Hamiltonian cycle. There is currently no efficient (i.e. quick) algorithm for solving this problem.

With the current best algorithms:

For a graph on 10 vertices, it takes on the order of 100 steps to check if it has a Hamiltonian cycle.

For 100 vertices, it takes on the order of \( 1,000,000,000,000,000,000,000,000 \) steps.
Complexity classes

We categorize our decision problems into different complexity classes which measure how efficiently (quickly) a problem can be solved. Today we will talk about 3 classes.

- **P**: all decision questions that can be solved efficiently (P stands for polynomial time)
  - Does a graph have an Eulerian circuit? Algorithm: check if the degree of every vertex is even. If yes: the graph has an Eulerian circuit. If no: it does not. This is an efficient algorithm!
  - Is a given number \( n \) prime? Algorithm: check if \( n \) is divisible by any number from 2 to \( n-1 \). If yes: \( n \) is not prime. If no: \( n \) is prime. Actually we only need to check the numbers from 2 to \( \sqrt{n} \).
Complexity classes

- NP: decision questions such that if the answer is ‘yes’, then there is an efficient way to check if a proposed solution is correct.
  - Sudoku: given a “solution” to a specific Sudoku puzzle, we just need to check that there are no repeated numbers in each row, column, and block. This can be done quickly.
  - Hamiltonian cycles

If a decision problem is NP, solutions can be checked efficiently, but there may not be an efficient way to find a solution!

NP stands for “nondeterministic polynomial time”

- NP-complete: questions that are at least as hard as all other questions in NP.
  - An efficient algorithm for solving an NP-complete problem can be used to build an efficient solution to any other problem in NP.
Reductions

If you know how to add numbers then you know how to multiply (whole) numbers.

\[ 9 \times 5 = 9 + 9 + 9 + 9 + 9 \]

Similarly, if we know how to solve an NP-complete problem then we know how to solve any other problem in NP without much extra effort.
NP-complete questions

- Checking for Hamiltonian cycles in graphs
- Given some numbers, does there exist any set of $k$ of the numbers that adds up to 100?
- Boolean satisfiability ($k$-SAT) which has applications for protein folding and drug design

All of these problems are NP: that is, efficient to check solutions. But no known efficient algorithms exist to solve them.
Are there problems that are in both P and NP-complete?

We don’t know (right now). If there was a decision problem that was both P and NP-complete…

- We would have an efficient way to solve this problem (definition of P)
- But using this efficient solution, we can find an efficient solution for any other problem in NP (definition of NP-complete)
- Therefore all NP problems would also be P!
NP problems = P problems = NP-complete problems

Question: P = NP?

- If this were true, there would be monumental developments in biotechnology.
- Also, internet security would break down.
Proving P = NP or P ≠ NP

- There is currently a $1,000,000 prize for proving either P = NP or P ≠ NP.
- How to prove P = NP: show that there is an efficient algorithm for solving any 1 problem in NP-complete.
- Proving P ≠ NP is much harder! Unfortunately most people believe P ≠ NP.