DSC 40B

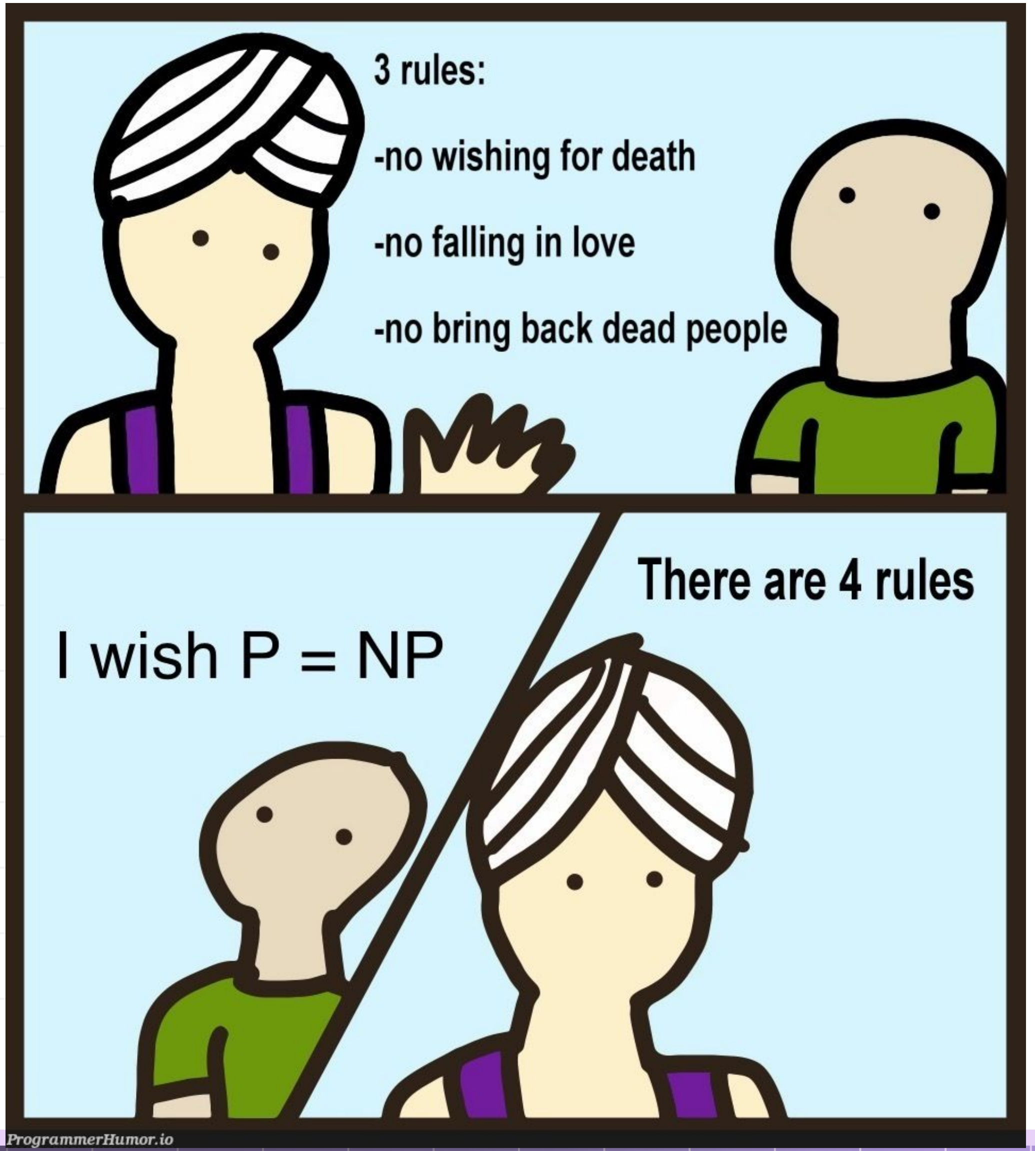
Lecture 29 :

Complexity Theory

Complexity Theory

Makes sense?

Funny?



The quest for efficient algorithms is about finding clever ways to avoid taking exponential time. So far we have seen the most brilliant successes of this quest; now we meet the quest's most embarrassing and persistent failures.

Paraphrased from *Algorithms* by Dasgupta, Papadimitriou, Vazirani

Exponential to Polynomial

- Many problems have brute force solutions which take **exponential time.**
- **Example**: Clustering to maximize separation
- The challenge of algorithm design: find a more **efficient solution**.

Polynomial Time

- If an algorithm's worst case time complexity is $O(n^k)$ for some k, we say that it runs in **polynomial time.**
    - **Example**: $\Theta(n \log n)$, since $n \log n = O(n^2)$.
    - **Polynomial** is much faster than **exponential** for big n.
- But not necessarily for small n.
    - **Example**: n^{100} vs 1.0001^n.
- We therefore think of polynomial as "efficient".

Question

- Is ***every*** problem solvable in polynomial time?

Question

- Is ***every*** problem solvable in polynomial time?
- **No! Problem**: print *all* permutations of n numbers.

A: n

B: 2^n

C: n!

D: Something else

Question

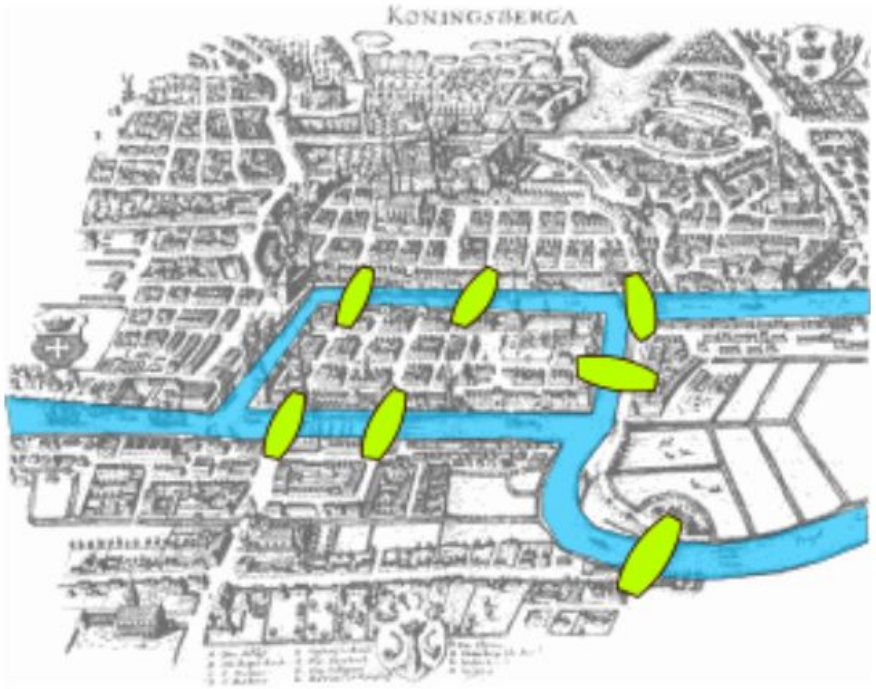
- Is ***every*** problem solvable in polynomial time?
- **No!** Problem: print *all* permutations of n numbers.
- **No! Problem**: given $n \times n$ checkerboard and some configuration of black and white pieces, determine if black can force a win.
  - Takes at least **exponential** time (**was proven**).

Ok, then...

- What problems can be solved in polynomial time?
- What problems can't?
- How can I tell if I have a hard problem?
- Core questions in **computational complexity theory**.
  - In a way, these questions are the foundation of computer science!

Eulerian and Hamiltonian Cycles

Example: Seven Bridges of Königsberg



Problem: Is it possible to start and end at same point while crossing each bridge **exactly once**?

Oiler

Leonhard Euler: 1707 - 1783

Oiler

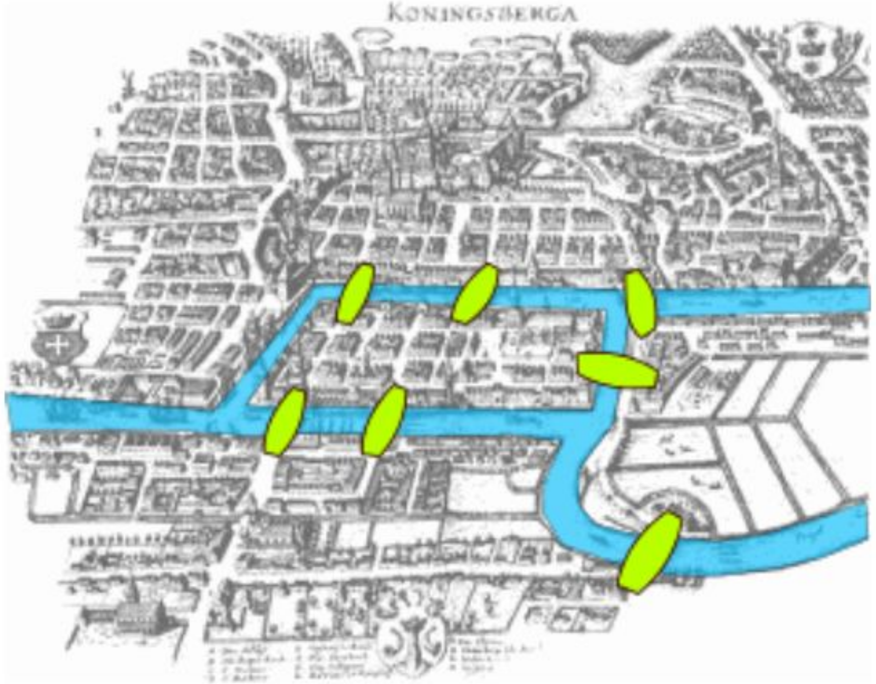
Leonhard Euler: 1707 - 1783

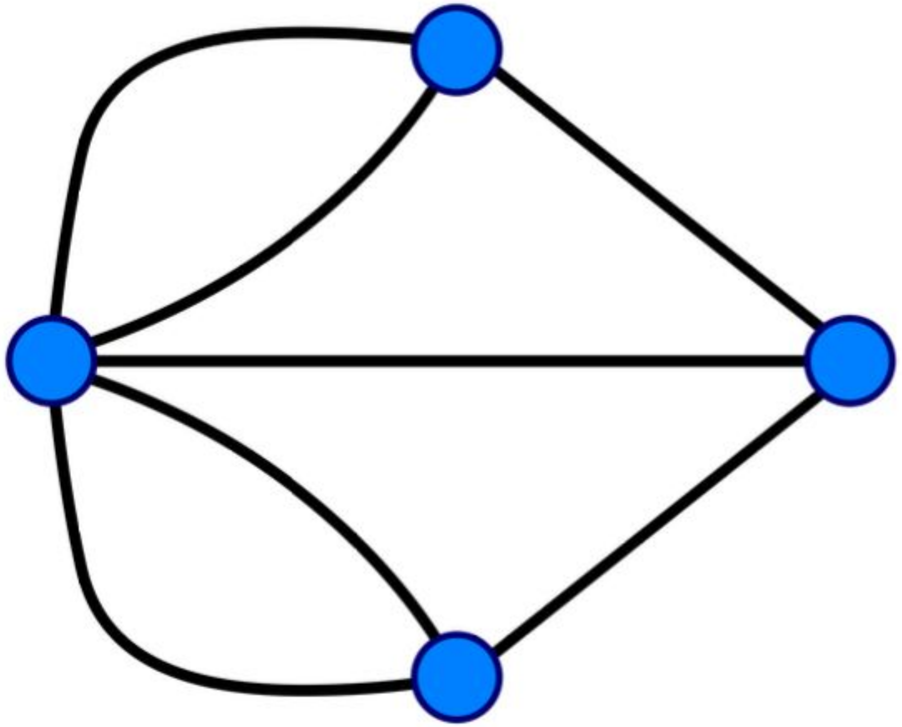
Side note:

e = 2.71828 is sometimes called *Euler's number.*

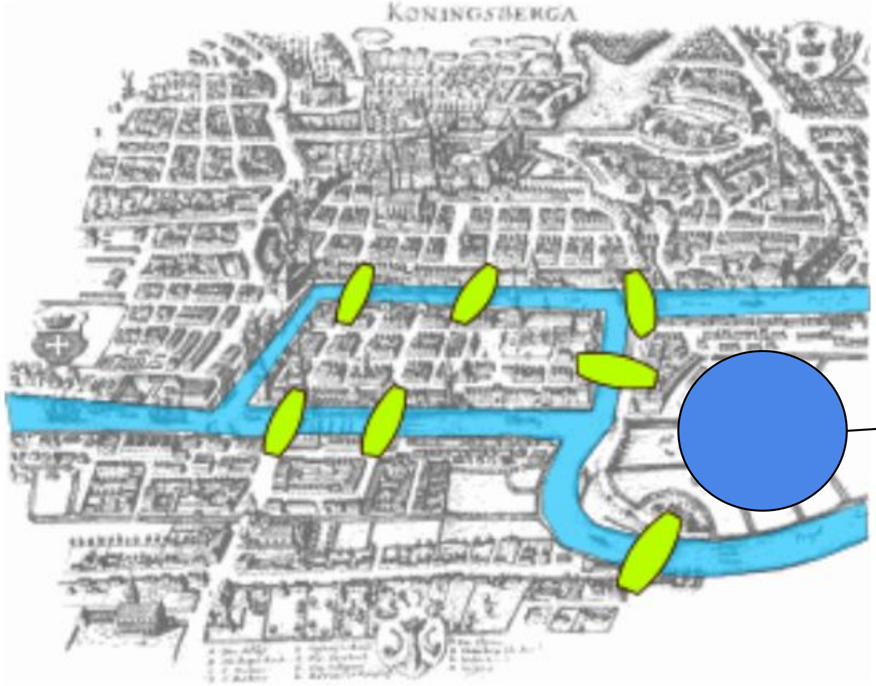
Also called *Napier's constant*

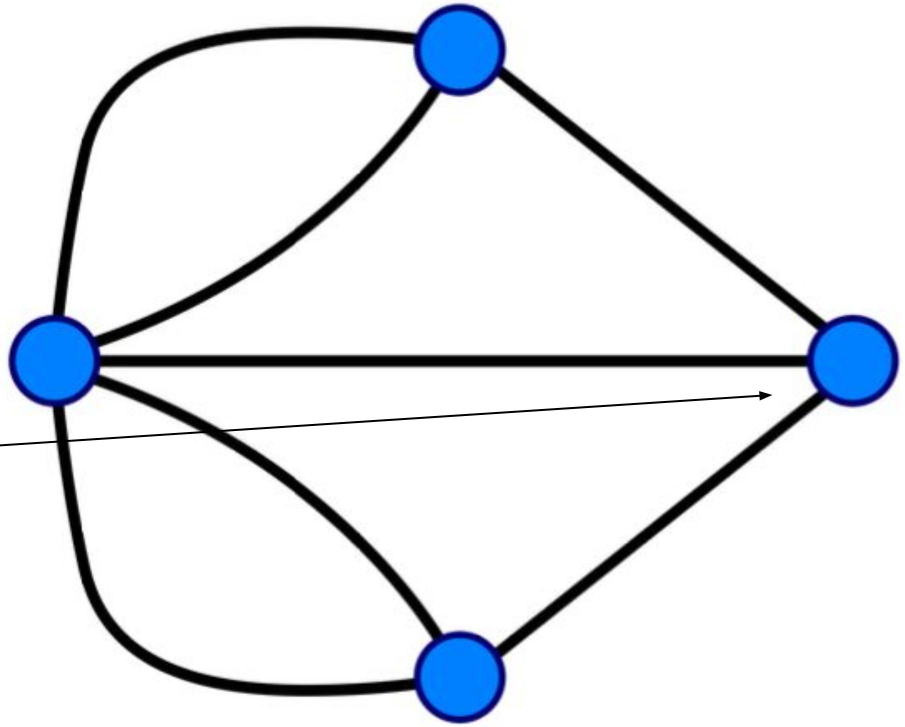
Eulerian Cycle



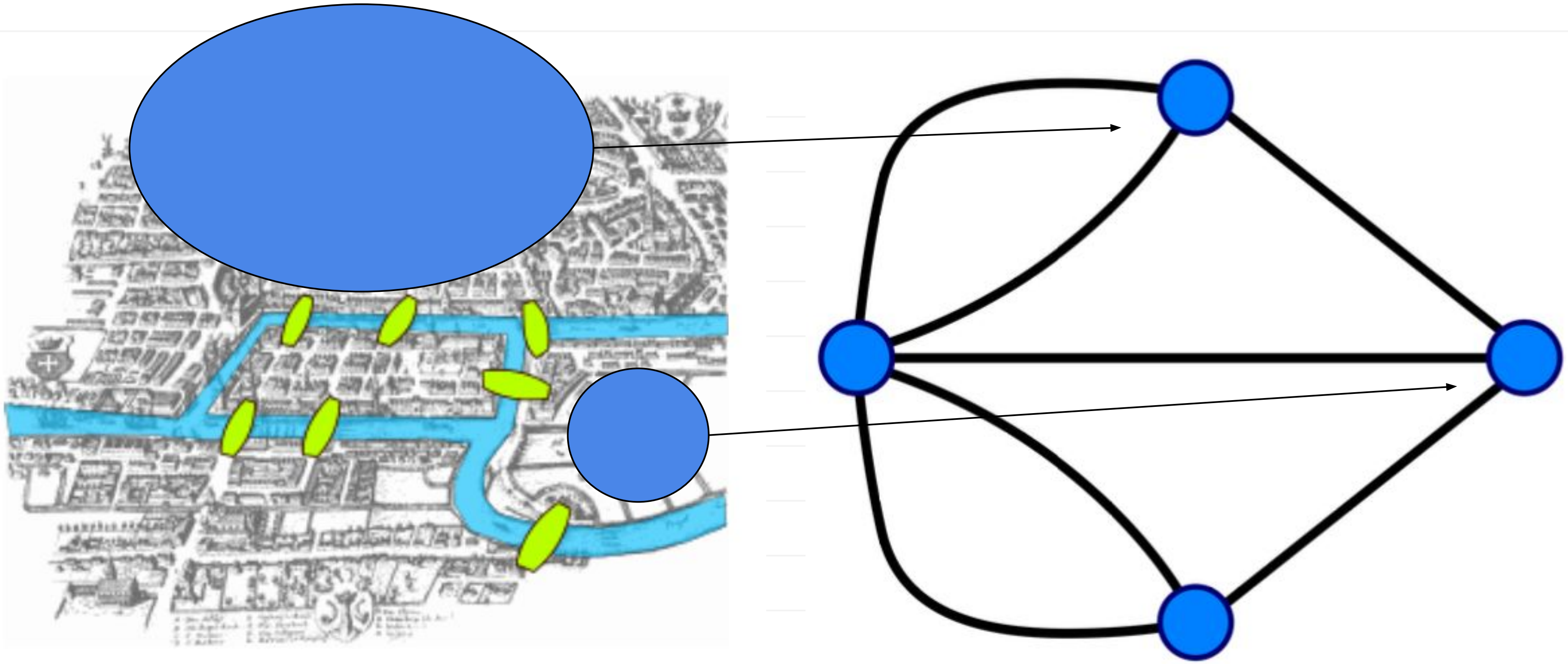


Eulerian Cycle

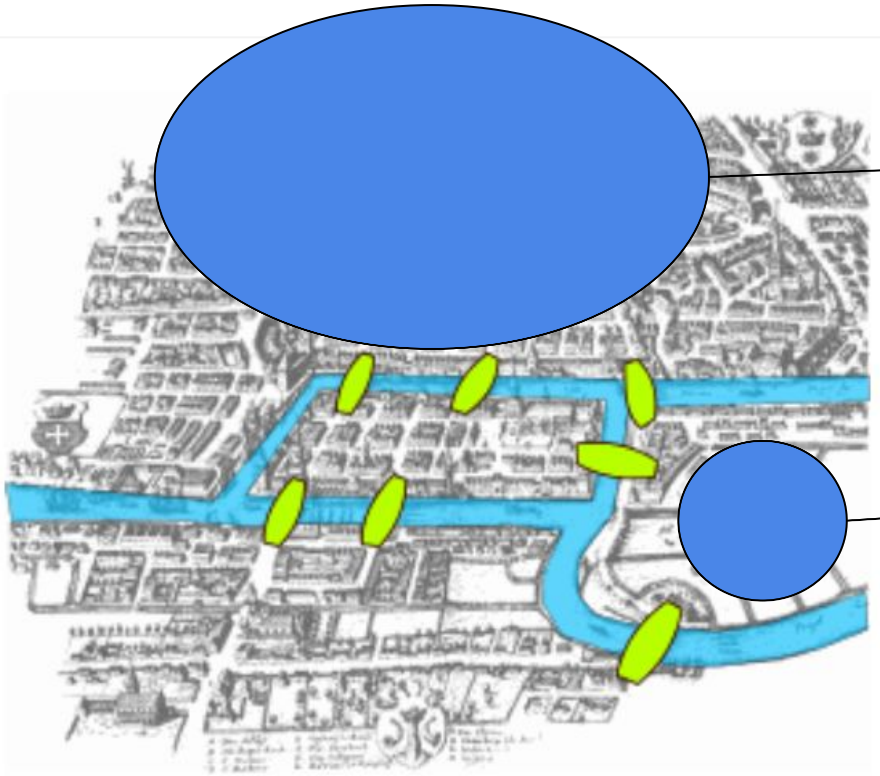


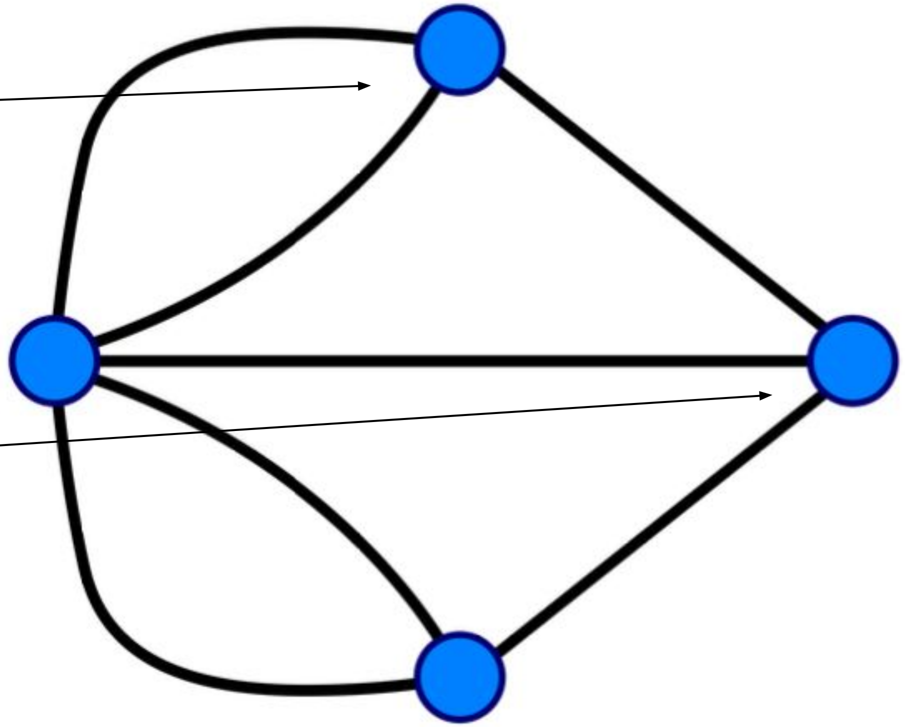


Eulerian Cycle

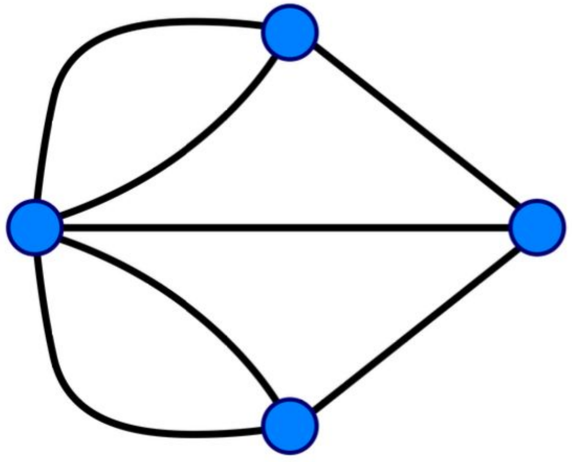


Eulerian Cycle: Is there a cycle which uses **each edge exactly once ?**





Necessary conditions



- Graph must be **connected**.
- Each node must have **even** degree.
- Answer for Königsberg problem: it is impossible.

In General...

- These conditions are **necessary** and **sufficient**.
- A graph has a Eulerian cycle **if and only if**:
  - it is connected;
  - each node has even degree.

Exercise

Can we determine if a graph has an Eulerian cycle in time that is polynomial in the number of nodes?

Oilerian

Answer

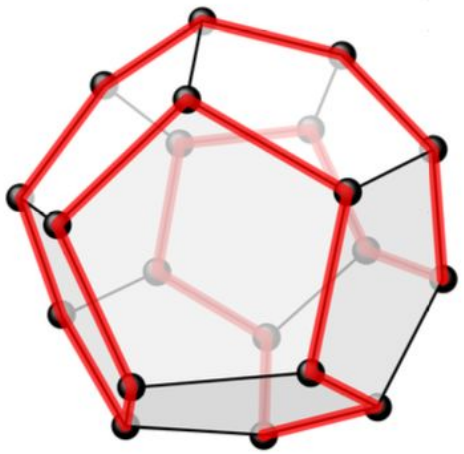
- We can check if it is connected in $\Theta(V + E)$ time.
  - **Q**: How?
  - **A**: DFS
- Compute every node's degree in $\Theta(V)$ time with adjacency list.
- Total: $\Theta(V + E) = O(V^2)$. **Yes!**

Gaming in the 19th Century

I have found that some young persons have been much amused by trying a new mathematical game which the Icosian furnishes [...]

- W.R. Hamilton, 1856

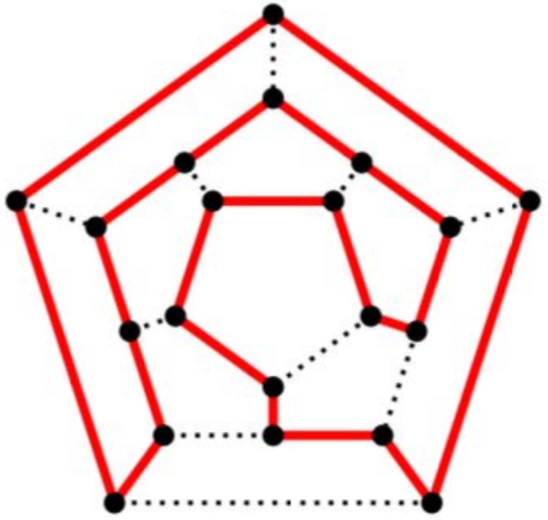
Dodecahedron



12 pentagons

Hamiltonian Cycles

- A **Hamiltonian cycle** is a cycle which visits **each** node exactly **once** (except the starting node).
- **Game**: find a Hamiltonian cycle on the graph below:



Exercise

Can we determine whether a general graph has a Hamiltonian cycle in polynomial time?

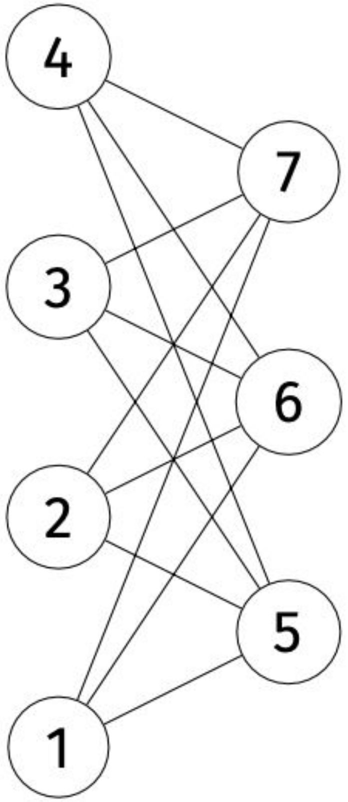
Fun Story (unrelated)

In the 1930s, George Dantzig was a graduate student at UC Berkeley. One day, he arrived late to class and saw two math problems written on the blackboard. Thinking they were a homework assignment, he copied them down and solved them over the next few days.

Later, he found out that:

- These problems were not homework.
- They were **open problems** — previously unsolved in the field of statistics.
- His solutions were actually **original breakthroughs**.

Some cases are easy: bipartite graph



In General

- Could brute-force.
- How many possible cycles are there?

In General

- Could brute-force.
- How many possible cycles are there?

In a complete undirected graph with |V| vertices:

- The number of Hamiltonian cycles is $(|V|-1)!/2$

Hamiltonian Cycles are Difficult

- This is a **very difficult** problem.
- No polynomial algorithm is known for general graphs.
- In special cases, there may be a fast solution.
- But in general, **worst case is hard**.

Note

- Determining if a graph has a Hamiltonian cycle **is hard.**
- But if we're given a "hint" (i.e., $(v_1, v_2, \ldots, v_n)$ is possibly a Hamiltonian cycle), we can check it **very quickly**!
- Hard to solve; but easy to verify "hints".

Similar Problems

- **Eulerian**: polynomial algorithm, "easy".
- **Hamiltonian**: no polynomial algorithm known, "hard".

Main Idea

Computer science is littered with pairs of similar problems where one easy and the other very hard.

Shortest and Longest Paths

- **Input**: Graph* G, source u, dest. v, number k.
- **Problem**: is there a path from u to v of length ≤ k?

Is there a solution to this problem?

A: Yes
B: No
C: May, not sure

* *Unweighted or weighted with no negative cycles*

Shortest and Longest Paths

- **Input**: Graph* G, source u, dest. v, number k.
- **Problem**: is there a path from u to v of length ≤ k?
- **Solution**: BFS or Dijkstra/Bellman-Ford in polynomial time.
- **Easy**!

* *Unweighted or weighted with no negative cycles*

Problem: LongPath

- **Input**: Graph G, source u, dest. v, number k.
- **Problem**: is there a **simple** path (visit a vertex once) from u to v of length ≥ k?
- **Naïve solution**: try all $V\,!$ path candidates.

Weighted or unweighted.

LongPaths

- There is **no** known polynomial algorithm for this problem.
- It is a **hard problem.**
- But given a “hint” (a possible long path), we can verify it very quickly!

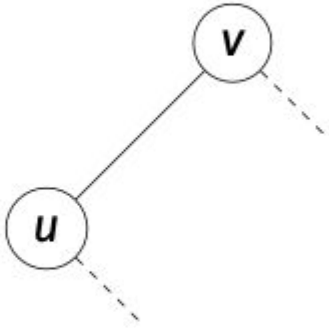
Reductions

Reductions

- `Hamiltonian` and `LongPath` are **related**.
- We can "convert" `Hamiltonian` into `LongPath` in polynomial time.
- We say that `Hamiltonian` **reduces** to `LongPath`.
    - To solve `Hamiltonian` I can solve `LongPath` and convert answer back to `Hamiltonian`.

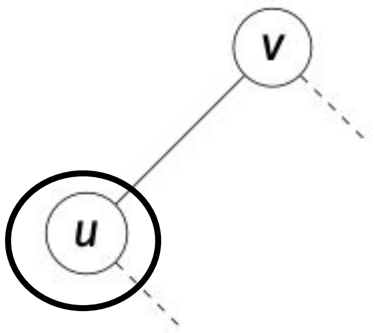
Reduction

- **Suppose we have an algorithm for LongPath.**
- We can use it to solve `Hamiltonian` as follows:



Reduction

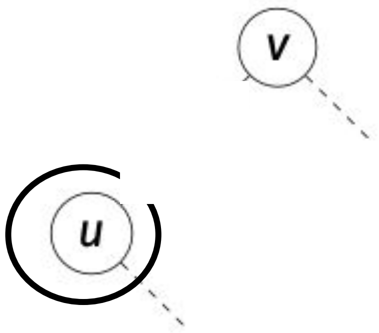
- Suppose we have an algorithm for `LongPath`.
- We can use it to solve `Hamiltonian` as follows:



- Pick **arbitrary** node u.

Reduction

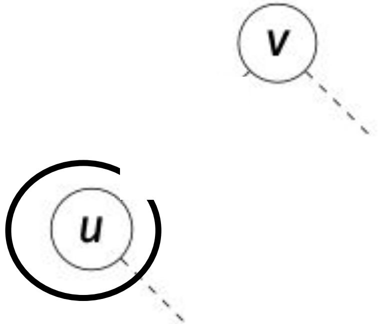
- Suppose we have an algorithm for `LongPath`.
- We can use it to solve `Hamiltonian` as follows:



- Pick **arbitrary** node u.
  - For each neighbor v of u:
    - Create graph G' by copying G, deleting (u, v)

Reduction

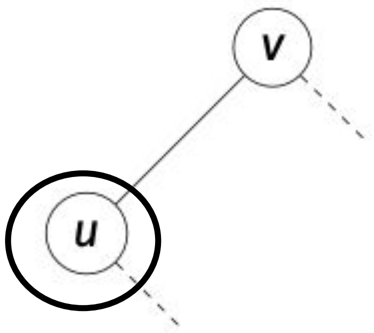
- Suppose we have an algorithm for `LongPath`.
- We can use it to solve `Hamiltonian` as follows:



- Pick **arbitrary** node u.
  - For each neighbor v of u:
    - Create graph G' by copying G, deleting (u, v).
    - Use algorithm to check if a simple path of length $\geq |V| - 1$ from u to v exists in G' (`LongPath` is used here)

Reduction

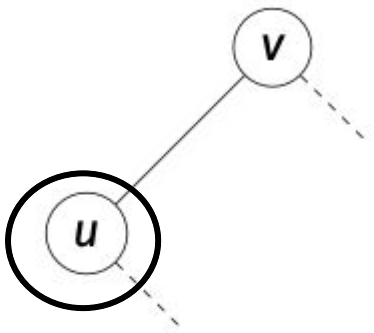
- Suppose we have an algorithm for `LongPath`.
- We can use it to solve `Hamiltonian` as follows:



- Pick **arbitrary** node u.
  - For each neighbor v of u:
    - Create graph G' by copying G, deleting (u, v)
    - Use algorithm to check if a simple path of length $\geq |V| - 1$ from u to v exists in G' (`LongPath` is used here)
    - If yes, then there is a Hamiltonian cycle (add a missing edge back)

Reduction

- Suppose we have an algorithm for `LongPath`.
- We can use it to solve `Hamiltonian` as follows:



- Pick **arbitrary** node u.
  - For each neighbor v of u:
    - Create graph G' by copying G, deleting (u, v)
    - Use algorithm to check if a simple path of length $\geq |V| - 1$ from u to v exists in G' (`LongPath` is used here)
    - If yes, then there is a Hamiltonian cycle (add a missing edge back)

Note: extra work is only polynomial.

Reductions

- If Problem A (`Hamiltonian`) reduces to Problem B (`LongPath`), it means "we can solve A by solving B".
- Best possible time for A ≤ best possible time for B + polynomial
  - Best time for `Hamiltonian` ≤ best for `LongPath` + polynomial
- "A is no harder than B".
- "B is at least as hard as A".

Informally:

A ≤ B

We'll assume reduction takes polynomial time.

Reductions

- If Problem A **reduces** (We'll assume reduction takes polynomial time) to Problem B, it means "we can solve A by solving B".
- Best possible time for A ≤ best possible time for B + polynomial
- "A is no harder than B"
- "B is at least as hard as A"

Relative Difficulty

- If Problem A reduces to Problem B, we say B is **at least as hard** as A.
- **Example**: `Hamiltonian` reduces to `LongPath`.

  `LongPath` **is at least as hard** as `Hamiltonian`.

P ?= NP

Decision Problems

- All of today's problems are decision problems.
    - **Output**: yes or no.
    - **Example**: Does the graph have an Euler cycle?

P: Polynomial

- Some problems have polynomial time algorithms.
  - `ShortPath`, `Euler`
- The set of decision problems that can be solved in polynomial time is called **P**.
- **Example**: `ShortPath` and `Euler` are in P.

NP: Non-deterministic polynomial

- The set of decision problems with "hints" that can be verified in polynomial time is called **NP**.
- All of today's problems are in NP.
  - All problems in P are also in NP.
- **Example**: `ShortPath`, `Euler`, `Hamiltonian`, `LongPath` are all in NP.

More Examples of NP problems

- Can you **drive across** the given set of cities, return back without running of gas.
  - Hard to solve, but easy to verify!
- **Scheduling.** Given set of courses and you need to schedule exams during the finals week such that no student takes two different exams on the same day.
- **All prime factors of a number**.
  - Easy to verify: prime factors of 234024 are 2, 3, 7, 199

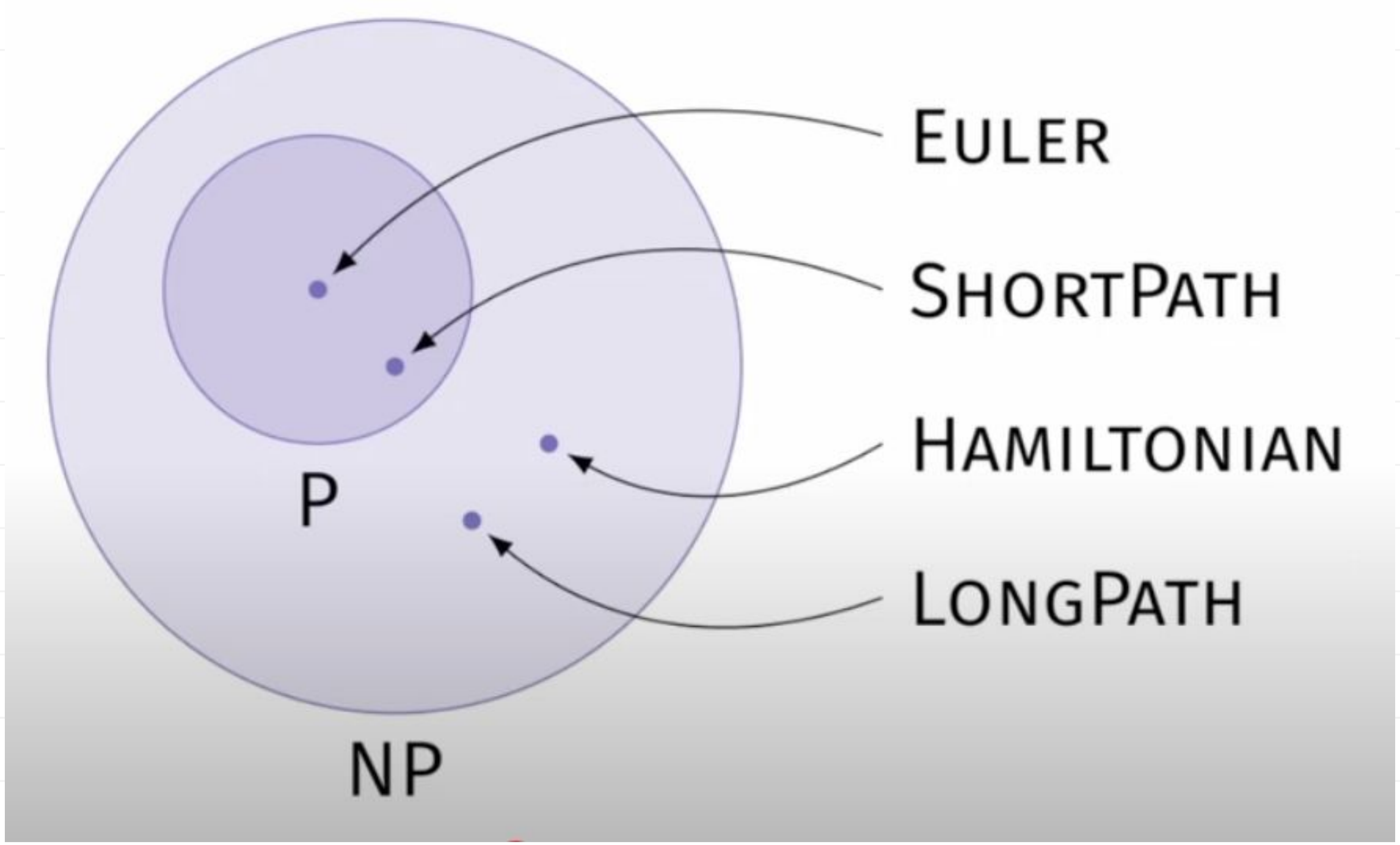
$P \subset NP$

- P is a subset of NP.
- It ***seems*** like some problems in NP aren't in P.
  - **Example**: `Hamiltonian`, `LongPath`.
- We don't know polynomial time algorithms for these problems.
- But that doesn't mean such an algorithm is impossible!

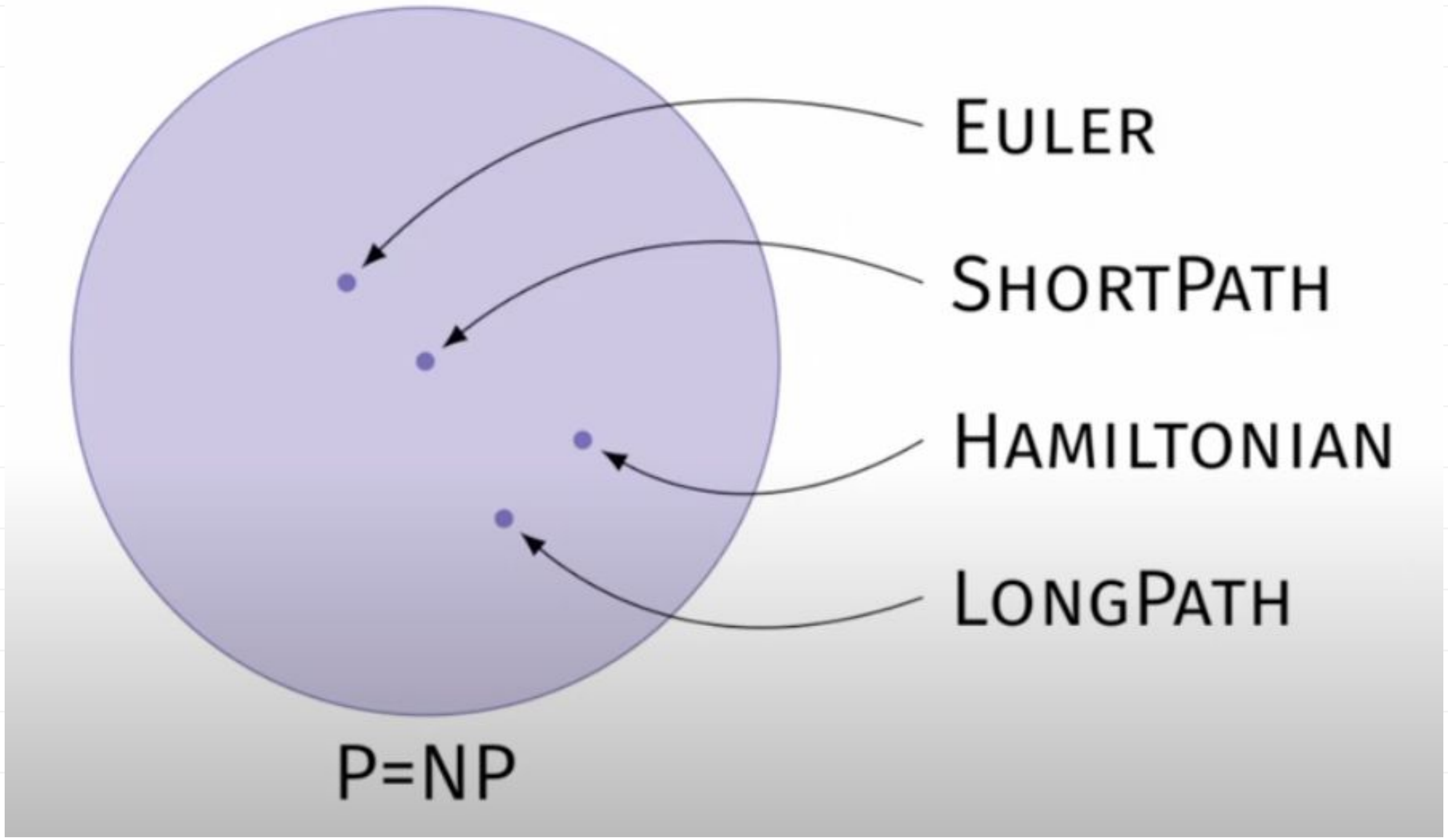
P = NP?

- Are there problems in NP that aren't in P?
  - **That is, is $P \neq NP$?**
- Or is any problem in NP also in P?
  - **That is, is $P = NP$?**

P ≠ NP: maybe?



P = NP: maybe?



P = NP?

- Is P = NP?
- **No one knows!**
- Biggest open problem in Math/CS.
- **Most think P ≠NP.**

If you solve it, you'll be rich and famous.

These seven problems were selected by the Clay Mathematics Institute (CMI) in 2000, and the CMI has offered a US$1 million prize for the first correc solution to each.

The seven Millennium Prize Problems are:

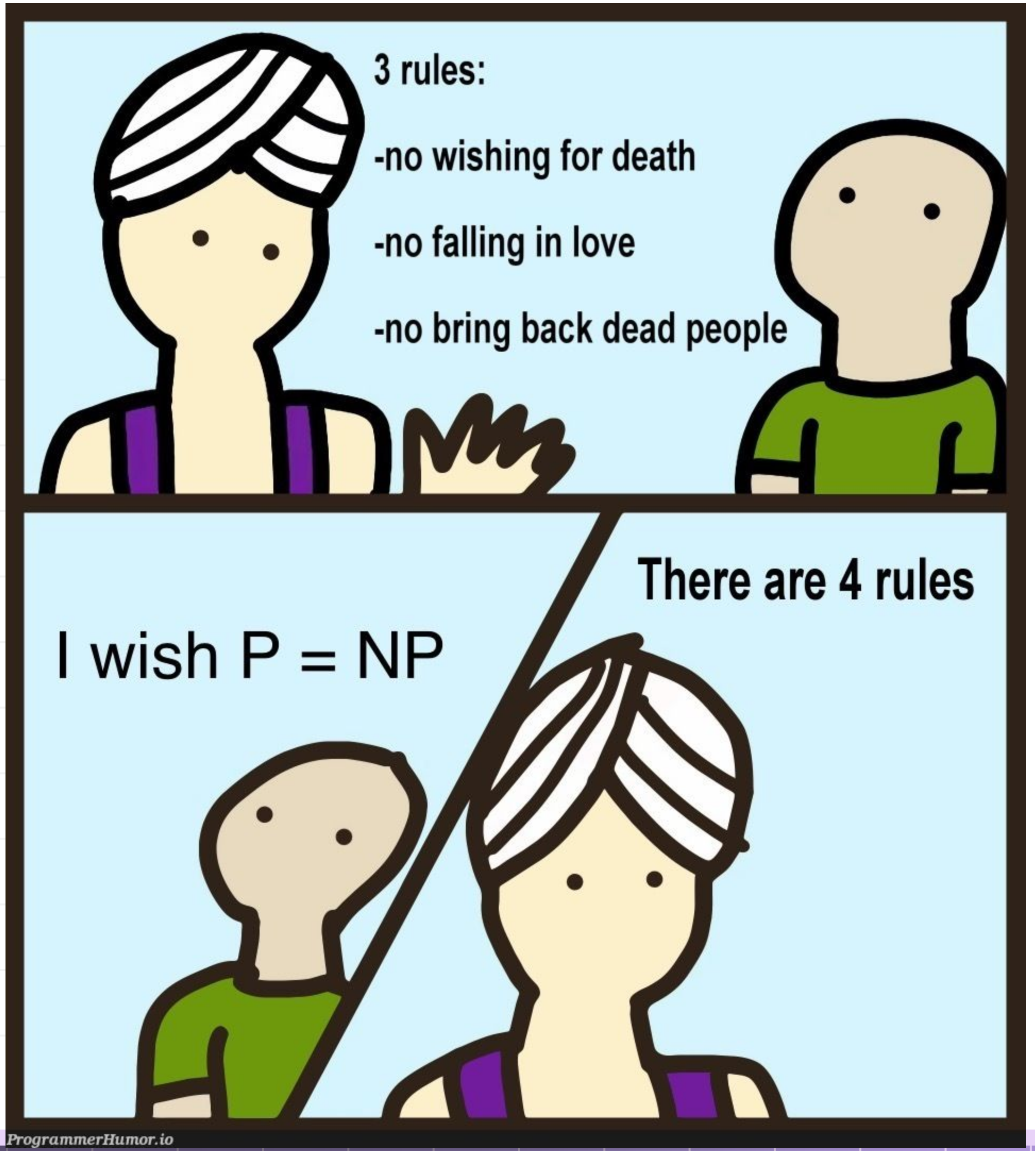
- **Yang-Mills Existence and Mass Gap:** This problem concerns the existence and properties of quantum Yang-Mills theories, which are fundamental to the Standarc Model of particle physics.
- **Riemann Hypothesis:** This hypothesis, formulated in 1859, concerns the distribution of prime numbers and the zeroes of the Riemann zeta function.
- **P versus NP problem:** This problem asks whether every problem whose solution can be quickly verified can also be quickly solved.
- **Navier–Stokes Existence and Smoothness:** These equations describe the motior of fluids like water and air. The problem asks whether solutions to these equations always exist and are smooth (well-behaved).
- **Hodge Conjecture:** This conjecture relates algebraic geometry to the topology of complex manifolds.
- **Poincaré Conjecture:** (Solved by Grigori Perelman) This conjecture, proposed by Henri Poincaré in 1904, stated that any simply connected, closed 3-manifold is equivalent to a 3-sphere.
- **Birch and Swinnerton-Dyer Conjecture:** This conjecture relates the number of points on an elliptic curve over a finite field to the behavior of a related function.

So far, only the Poincaré Conjecture has been solved, by Grigori Perelman in 2003. He declined the prize money.

What if P = NP?

- **Possibly Earth-shattering.**
  - Almost all cryptography instantly becomes obsolete;
  - Logistical problems solved exactly, quickly;
  - ...
- **But maybe not…**
  - Proof could be non-constructive.
  - Or, constructive but really inefficient. E.g., $\Theta(n^{10000})$

Does it make sense now?



NP-Completeness

Problem: 3-SAT

- Suppose x_1, … , x_n are boolean variables **(True,False)**
- A 3-clause is a combination made by or-ing and possibly *negating* three variables:
  - x_1 or x_5 or (not x_7)
  - (not x_1) or (not x_2) or (not x_4)

Problem: 3-SAT

- **Given**: m clauses over n boolean variables.
- **Problem**: Is there an assignment of x_1, … , x_n which makes all clauses **true** simultaneously?
- **No polynomial** time algorithm is known.
- But it is easy to verify a solution, given a hint.
  - 3-SAT is in NP.

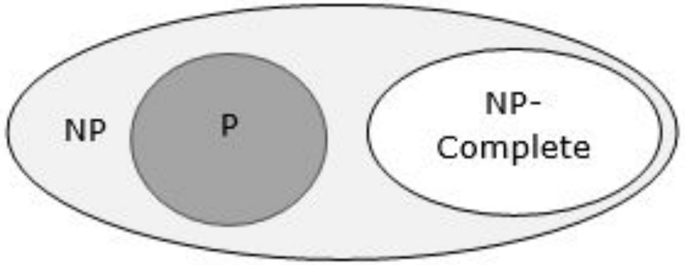
Cook's Theorem:

- **Every** problem in NP is polynomial-time **reducible** to 3-SAT.
  - ...including `Hamiltonian`, `LongPath`, etc.
  - 3-SAT is **at least as hard** as every problem in NP.
  - "hardest problem in NP"

Cook's Theorem (Corollary)

- If 3-SAT is solvable in polynomial time, then all problems in NP are solvable in polynomial time.
- ...including `Hamiltonian`, `LongPath`, etc.

NP-Completeness



- We say that a problem is **NP-complete** if:
  - it is in NP;
  - every problem in NP is *reducible* to it.
- `Hamiltonian`, `LongPath`, `3-SAT` are all **NP-complete**.
- NP-complete problems are the "hardest" in NP.

Equivalence

- In some sense, NP-complete problems are equivalent to one another.
- E.g., a fast algorithm for `Hamiltonian` gives a fast algorithm for `3-SAT`, `LongPath`, and all problems in NP.

Who cares?

- Complexity theory is a fascinating piece of science.
- But it’s practically useful, too, for recognizing hard problems when you stumble upon them.

Hard Optimization Problems

Hard Optimization Problems

- NP-completeness refers to **decision** problems.
- What about **optimization** problems?
- We can typically state a similar decision problem.
- If that decision problem is hard, then optimization is at least as hard.

Problem: bin packing

- Optimization problem:
  - **Given**: bin size B, n objects of size $\alpha_1, \ldots, \alpha_n$..
  - **Problem**: find minimum number of bins k that can contain all objects.
- Decision problem version:
  - **Given**: bin size B, n objects of size $\alpha_1, \ldots, \alpha_n$, integer k.
  - **Problem**: is it possible to pack all n objects into k bins?
- Decision problem is NP-complete, reduces to optimization problem.

Example: traveling salesperson

- Optimization problem:
  - **Given**: set of n cities, distances between each.
  - **Problem**: find shortest Hamiltonian cycle.
- Decision problem:
  - **Given**: set of n cities, distance between each, length ℓ.
  - **Problem**: is there a Hamiltonian cycle of length $\leq \ell$?
- Decision problem is NP-complete, reduces to optimization problem.

NP-complete problems in machine learning

- Many machine learning problems are NP-complete.
- **Examples**:
  - Finding a linear decision boundary to minimize misclassifications in non-separable regime.
  - Minimizing k-means objective.

So now what?

- Just because a problem is NP-Hard, doesn’t mean you should give up.
- Usually, an approximation algorithm is fast, “good enough”.
- Some problems are even hard to approximate.

Summary

- Not every problem can be solved efficiently.
- Computer scientists are able to categorize these problems.

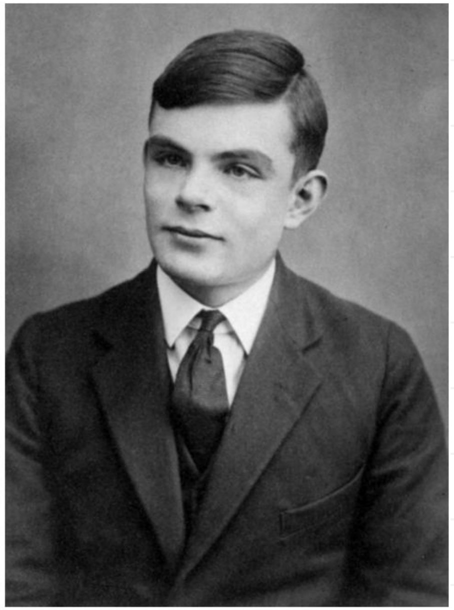
THE END

NP-Hard

Really hard problems

- Some decision problems are harder than others.
- That is, it takes more time to solve them.
- Given enough time, all decision problems can be solved, right?

Alan Turing: 1912-1954



Turing's Halting Problem

- **Given**: a function f and an input x.
- **Problem**: does f(x) halt, or run forever?
- Algorithm must work for all functions/inputs!

Turing's Argument

- Turing says: no such algorithm can exist.
- Suppose there is a function halts(f, x):
- Returns **True** if f(x) halts.
- Returns **False** if f(x) loops forever.

Turing's Argument

```python
def evil_function(f):
    if halts(f, f):
        # loop forever
    else: # it runs forever
        return
```

- Consider `evil_function(evil_function)`.
  - Does it halt or not?

Turing's Argument

```python
def evil_function(f):
    if halts(f, f):
        # loop forever
    else: # it runs forever
        return
```

- Consider `evil_function(evil_function)`.
    - Does it halt or not?
- Assuming that halt works leads to logical impossibility!
    - So a working halt cannot exist.

Undecidability

- The halting problem is **undecidable**.
- Fact of the universe: there can be no algorithm for solving it which works on all functions/inputs.
- All of these problems are undecidable:
    - Does the program terminate?
    - Does this line of code ever run?
    - Does this function compute what its specification says?
    - Many others...

Reality

- **Physics**: can’t go faster than the speed of light.
- **Computer science:**
  - There’s a speed limit for certain problems, too.
  - And some problems can’t even be solved!

Thank you!

Do you have any questions?

CampusWire!