P vs NP

CSE 332 Summer 2021

Instructor: Kristofer Wong

Teaching Assistants:

Alena Dickmann	Arya GJ	Finn Johnson
Joon Chong	Kimi Locke	Peyton Rapo
Rahul Misal	Winston Jodjana	

Announcements

- Almost done! Remaining Deadlines:
  - P3: Tuesday, late days: Thursday
    - TA's will be grading throughout the week - hope to release grades Friday around noon but this depends on how many of you submit Thursday at 11:59.
    - Regardless of when we are able to get grades out (sometime Friday), you will only have until noon Saturday to submit any regrade requests (They will open immediately on this one)
  - Final reflection: Thursday, 11:59

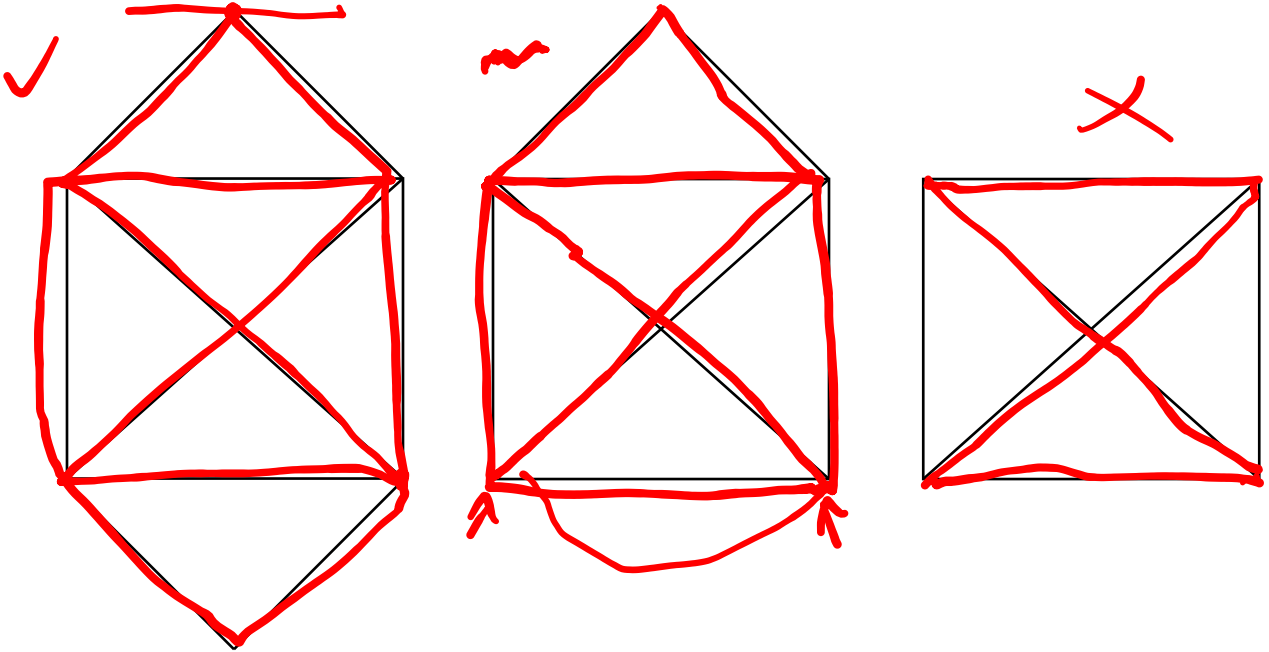
- Course Evals: https://uw.iasystem.org/survey/245272

Lecture Outline

- **Interesting Problems:**
  - Euler's Circuit
  - Hamiltonian Circuit
- What are P and NP?
- NP-Completeness
- Dealing with your enemy

Welcome to TikTok! Only 1% of people can do this 😰

- Draw (ie, trace all edges) these shapes without lifting your pencil, drawing each line only once!
  - Bonus points: Can you start and end at the same spot?



Wait, that was a graph problem?

- Imagine each point where lines intersect were nodes and lines were edges.

- A more realistic question, yet same problem:
  - Your company has to inspect a set of roads between cities by driving over each of them.
  - Driving over the roads costs money (fuel), and there are a lot of roads.
  - Your boss wants you to figure out how to *drive over each road exactly once,* returning to your starting point.

- Let's solve this for your boss!

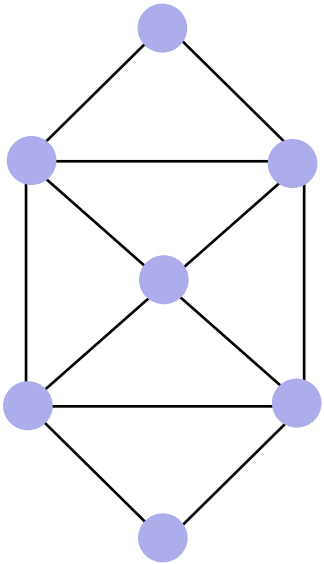
Euler Circuits

- Definition: a ~~path through~~ cycle in a graph that *visits each **edge** exactly once* (starting and ending at the same vertex)

- Named after Leonhard Euler (1707-1783), who cracked this problem and founded graph theory in 1736 (and a lot more)

- How to tell if the internet troll is bs'ing you
  - An Euler circuit exists *iff*
    - the graph is connected
    - each vertex has even degree (= # of edges on the vertex)

Euler Circuits: Solution



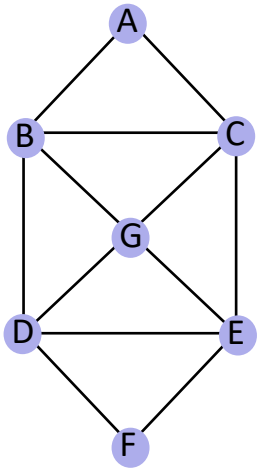
- Given a connected, undirected graph G = (V,E)

- Can check if a circuit exists:
  - Do all vertices have even degree? O(|V|)

- Can find a circuit:
  1. Traverse graph from start vertex until you are back
     - Never get stuck because of the even-degree property
  2. "Remove" the cycle, leaving several components each with the even-degree property
     - Recursively find Euler circuits for these
  3. Splice all these circuits into an Euler circuit

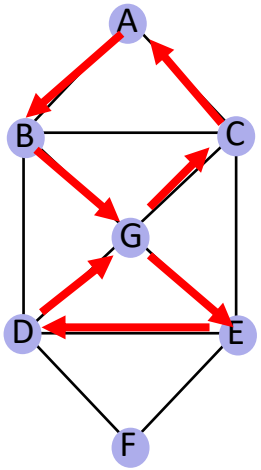
- Can verify a given path is a circuit:
  - Traverse path, marking visited edges
  - Return true if all edges are marked, and $v_0 == v_n$

Euler Circuit: Example



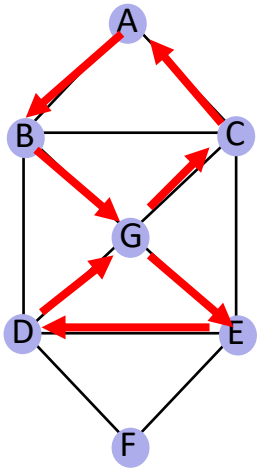
Euler(A):

Euler Circuit: Example

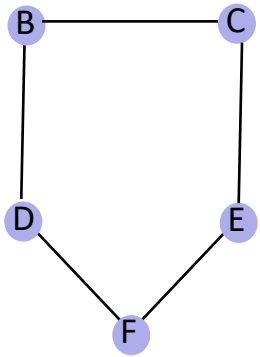


Euler(A):
A B G E D G C A

Euler Circuit: Example

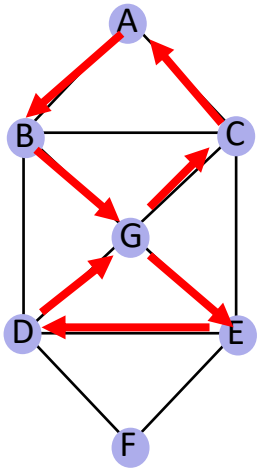


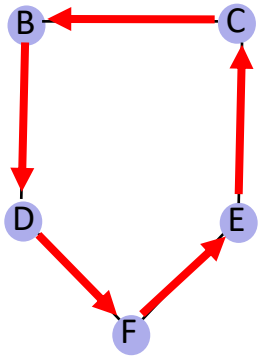
Euler(A):
A B G E D G C A



Euler(B):

Euler Circuit: Example





Euler(A):
A **B** G E D G C A

Euler(B):
B D F E C **B**

Spliced: A **B D F E C B** G E D G C A

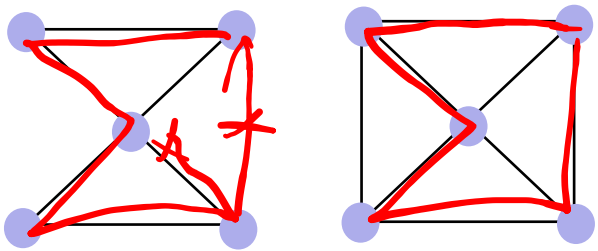
Another task

- Your boss is pleased... and therefore assigns you a new task
  - It's ok, you're paid hourly
- Your company has to send someone by car to a set of cities
- These cities all charge a hefty toll to enter each city
- Your boss wants you to minimize costs: figure out *how to drive to each city exactly once, returning in the end to the city of origin*.

Hamiltonian Circuits

- Definition: A cycle that goes through each *vertex* exactly once
  - Recall Euler's circuit: A cycle that goes through each *edge* exactly once

- Does the left graph have:
  - An Euler circuit? Y
  - A Hamiltonian circuit? N
- Does the right graph have:
  - An Euler circuit? N
  - A Hamiltonian circuit? Y



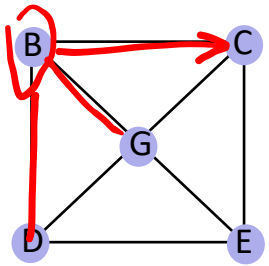
Finding Hamiltonian Circuits

- **Problem**: Find a Hamiltonian circuit in a connected, undirected graph G

- One solution: Search through *all paths* to find one that visits each vertex exactly once
  - Can use your favorite graph search algorithm to find paths
  - This is an *exhaustive search* ("brute force") algorithm

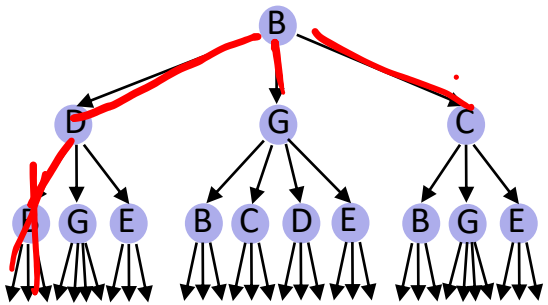
- Worst case: need to search all paths
  - **How many paths??**

Exhaustive Search: Analysis (1 of 2)



- Worst case: need to verify all paths
  - ***How many paths are there??***

- As with our lower-bound on comparison sorts, let's represent each step on a path as a node in a search tree
  - Number of leaves is the total number of paths

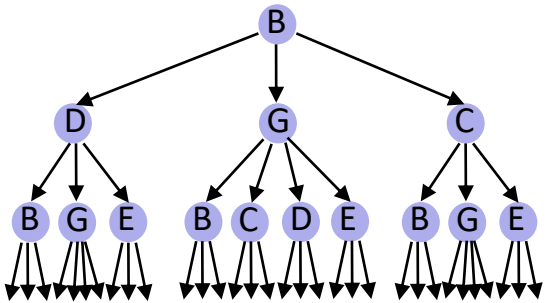


Search tree of paths starting at B

Exhaustive Search: Analysis (2 of 2)

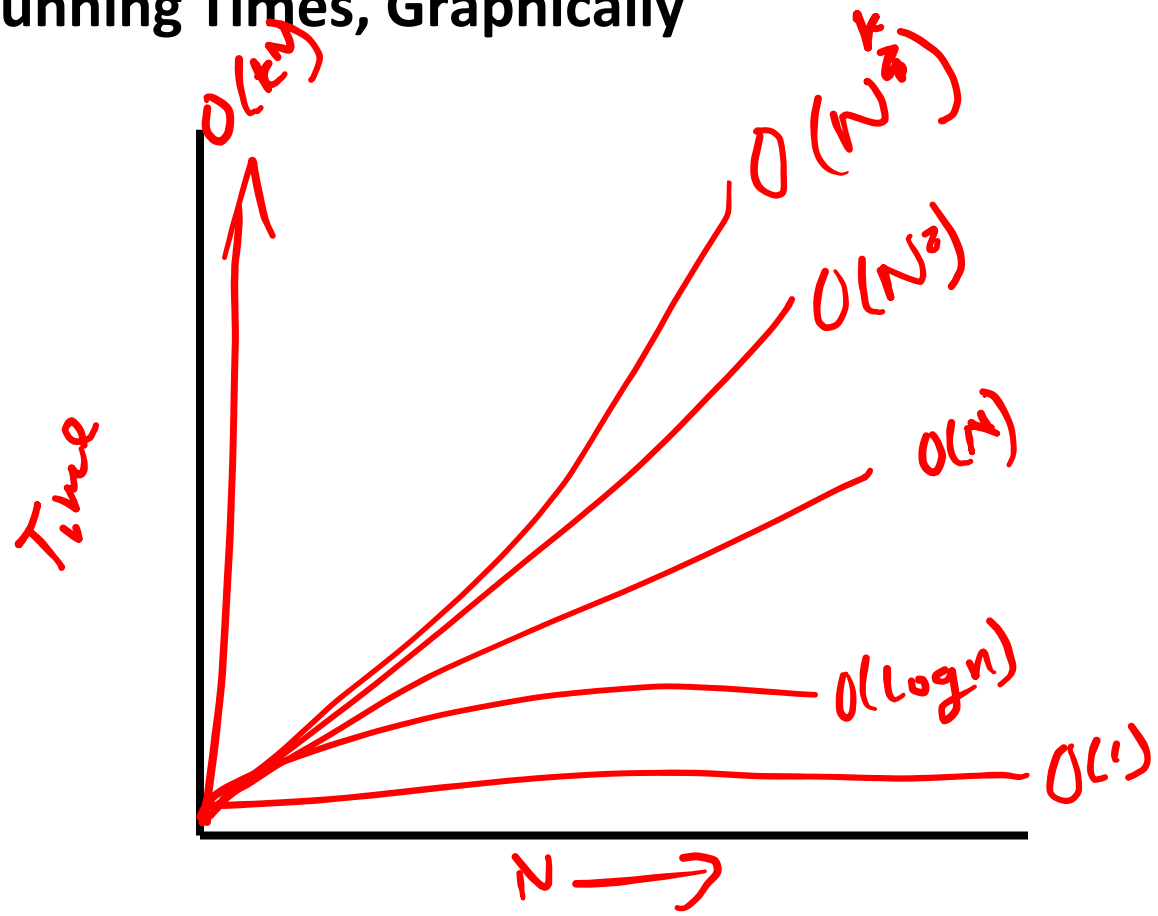
- Let *b* be the *average* branching factor of each node in this
  - |V| vertices, each with ~b branches
  - Total number of paths ~b·b·b … ·b
    - **O($b^{|V|}$)**

- Worst case:
  - ***Exponential time!***



Search tree of paths starting at B

Running Times, Graphically



More on Running Times

1.4×10^{10} < universe

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

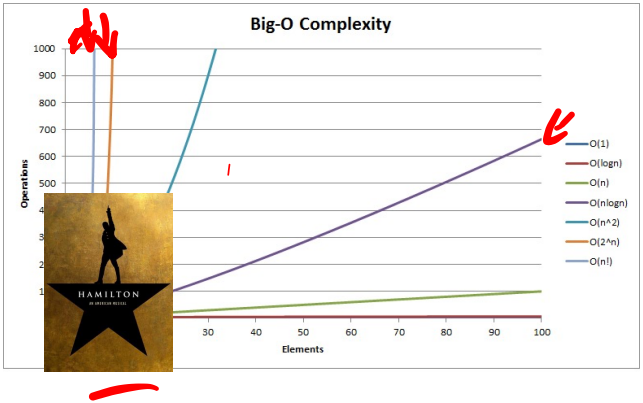
	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

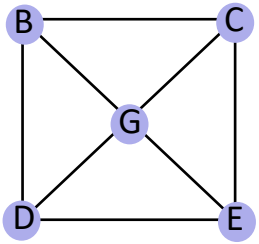
Somewhat old, from Rosen (old 311 textbook)

Summary: Euler vs Hamiltonian Circuits

- ***Euler circuit***: a cycle that goes through each *edge* exactly once
  - Runtime: O(|V| + |E|)

- ***Hamiltonian circuit***: a cycle that goes through each *vertex* exactly once
  - Runtime: $O(b^{|V|})$





Summary: Polynomial vs. Exponential Time

- All the algorithms we've discussed so far are ***polynomial time*** algorithms:
  - i.e.: algorithms whose running time is $O(N^k)$ for some $k > 0$
  - e.g.: $O(\log N)$, $O(N)$, $O(N \log N)$, $O(N^2)$, etc

- ***Exponential time*** algorithms run in $O(b^N)$ for some $b > 1$
  - Any exponential time algorithm is asymptotically worse than any polynomial function N^k
    - Holds true for any k and any b!
  - e.g.: $O(2^N)$

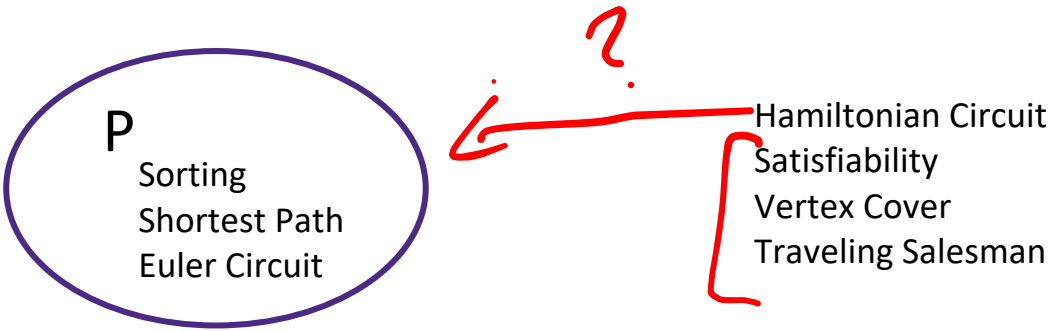
Lecture Outline

- Interesting Problems:
  - Euler's Circuit
  - Hamiltonian Circuit
- **What are P and NP?**
- NP-Completeness
- Dealing with your enemy

The Complexity Class P

- ***P*** is the set of all problems that can be solved in *polynomial worst-case time*
  - i.e.: all problems that have some algorithm with runtime $O(N^k)$

- Examples of problems in P:
  - Sorting, shortest path, Euler circuit, etc.


P
Sorting
Shortest Path
Euler Circuit
?
Hamiltonian Circuit
Satisfiability
Vertex Cover
Traveling Salesman

Probably not P: Satisfiability

- *Input*: a logic formula of size *m* containing *n* variables
  - e.g. $(\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee \neg x_5)$

2 2 2

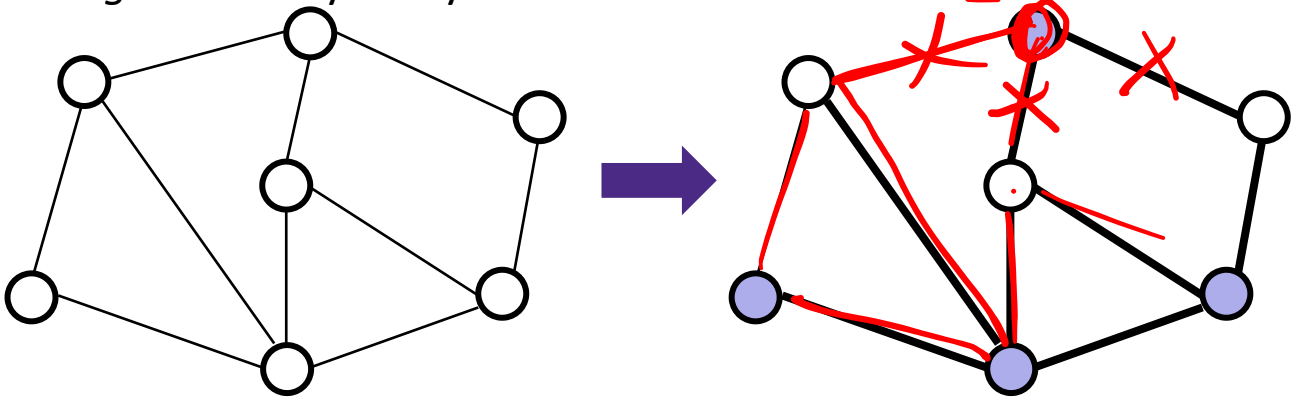
- *Output*: An assignment of boolean values to the *n* variables such that the formula is true

- *Algorithm*: Try every variable assignment

2^n

Probably not P: Vertex Cover

- *Input*: A graph G = (V,E) and a number *m*
- *Output*: A subset S of V, such that:
  - For every edge (u,v) in E, at least one of u or v is in S
  - |S|=*m* (if such an S exists)
- *Algorithm*: Try every subset of vertices of size *m*



Probably not P: Travelling Salesman

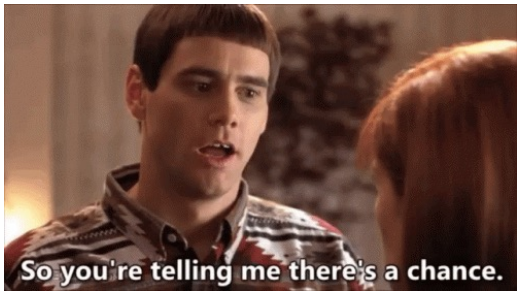
- *Input*: A complete weighted undirected graph G=(V,E) and a number *m*
- *Output*: A circuit visiting each vertex exactly once and has total cost <*m* (if such a circuit exists)
- *Algorithm*: Enumerate *all paths*, check if one of them is a circuit with appropriate weight

I said "Probably"

- If we have:
  - a *candidate solution* to a problem
  - the ability to *verify the solution in polynomial time*

  then maybe a polynomial-time algorithm exists?

- Does this hold true for the Hamiltonian Circuit problem?
  - Given a candidate path, how do we verify it's a Hamiltonian Circuit?
    - Check if all vertices are visited exactly once in the candidate path
    - Runtime: O(|V|)



The Complexity Class NP

- ***NP: "Non-Deterministic Polynomial"*** is the set of all problems for which a given candidate solution can be verified in *polynomial worst-case time*
  - Compare against ***P***, which are the problems that can be solved *in polynomial worst-case time*

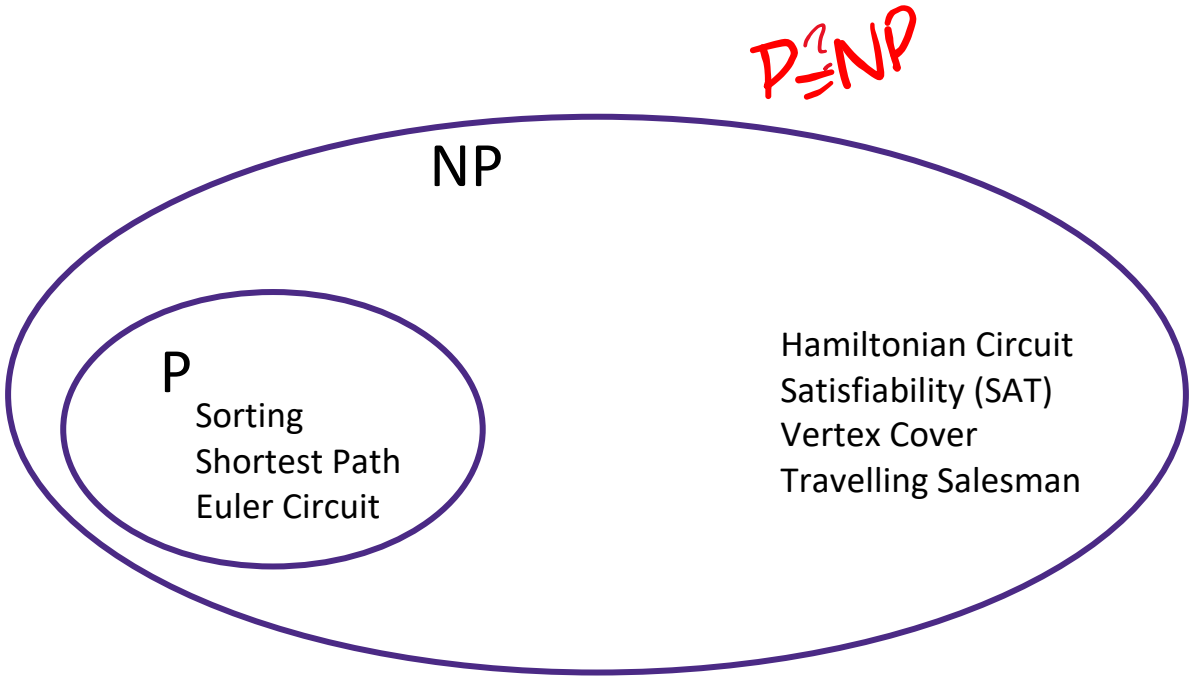
- Examples of problems in NP:
  - *Hamiltonian circuit*: Given a candidate path, can verify in O(|V|) time if it is a Hamiltonian circuit
  - *Satisfiability*: Given a candidate set of *n* values, can verify in O(*m*) time if the expression is true
  - *Vertex Cover*: Given a subset of vertices, can verify in O(|V|) time if it covers all vertices
  - *All problems that are in P* ***(why???)***

Why do we call it "NP"?

- NP stands for ***Nondeterministic Polynomial*** time
  - Unlike P, these problems are characterized by their *verification time*
  - Allows us to assume a solution exists (regardless of its runtime)

- Why "nondeterministic"?
  - If we don't know a polynomial time solution (yet?), we can still imagine a special operation that allows the algorithm to magically guess the right choice at each branch point
  - Nondeterministic algorithms don't exist – purely theoretical idea invented to understand how hard a problem could be

- "NP" is **NOT** an abbreviation for "not polynomial"


P?=NP
NP
P
Sorting
Shortest Path
Euler Circuit
Hamiltonian Circuit
Satisfiability (SAT)
Vertex Cover
Travelling Salesman

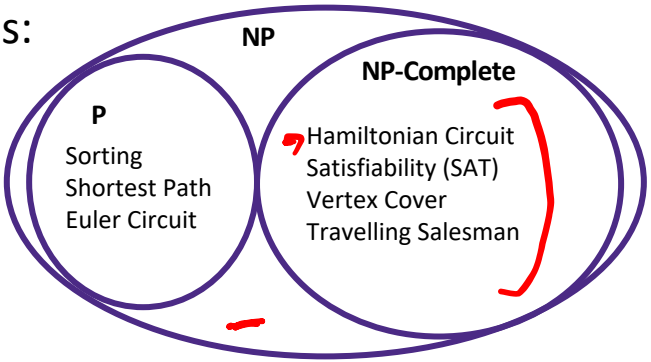
Who wants a 4.0… and a Turing Award? 🥺👉👈

- It is generally believed that P ≠ NP
  - i.e. there are problems in NP that are not in P

- But no one has been able to show even one such problem!
  - This is *the* fundamental open problem in theoretical computer science
  - Nearly everyone has given up trying to prove it. Instead, theoreticians prove theorems about what follows once we assume P ≠ NP !
    - Remember Richard Ladner?
      - Most famous for work in this area
      - Ladner's Theorem: https://en.wikipedia.org/wiki/NP-intermediate

The Complexity Class NP-Complete

- These are thought of as the **hardest** problems in the class NP.
- Special because any problem in NP can be "reduced" to any NP-complete problem in polynomial time
- **Interesting fact:** If any one NP-complete problem could be solved in polynomial time, then ***all*** NP-complete problems could be solved in polynomial time.
  - i.e. P = NP
- Example NP-complete problems:
  - Hamiltonian Circuit
  - Satisfiability
  - Vertex Cover
  - Travelling Salesman



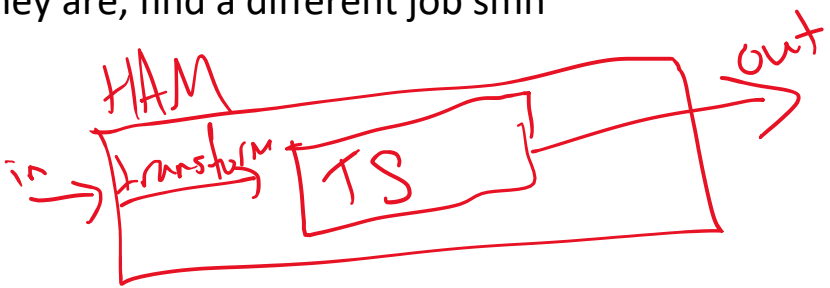
Our Enemy: One last time

- Turns out our enemy is the one who told our boss to give us the Hamiltonian circuit problem.
- Try as you might, every solution you come up with for the Hamiltonian Circuit problem runs in exponential time.....
- You have to report back to your boss…
- How do we keep our job?? Our options:
  - Keep working
  - Come up with an alternative plan...

In general, what to do with a Hard Problem

- If you can transform a known NP-complete problem into the one you're trying to solve in polynomial time, then you can stop working on your problem!

- If it's impossible (there is no known solution by **anyone**), your boss can't be upset with you
  - If they are, find a different job smh

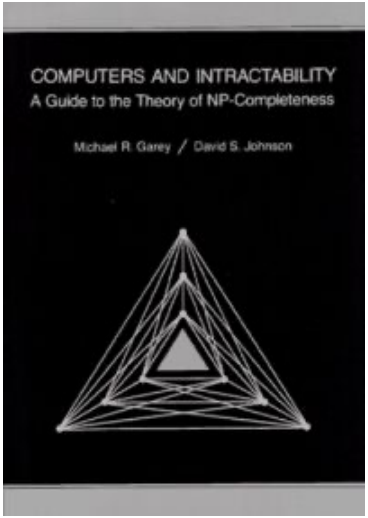


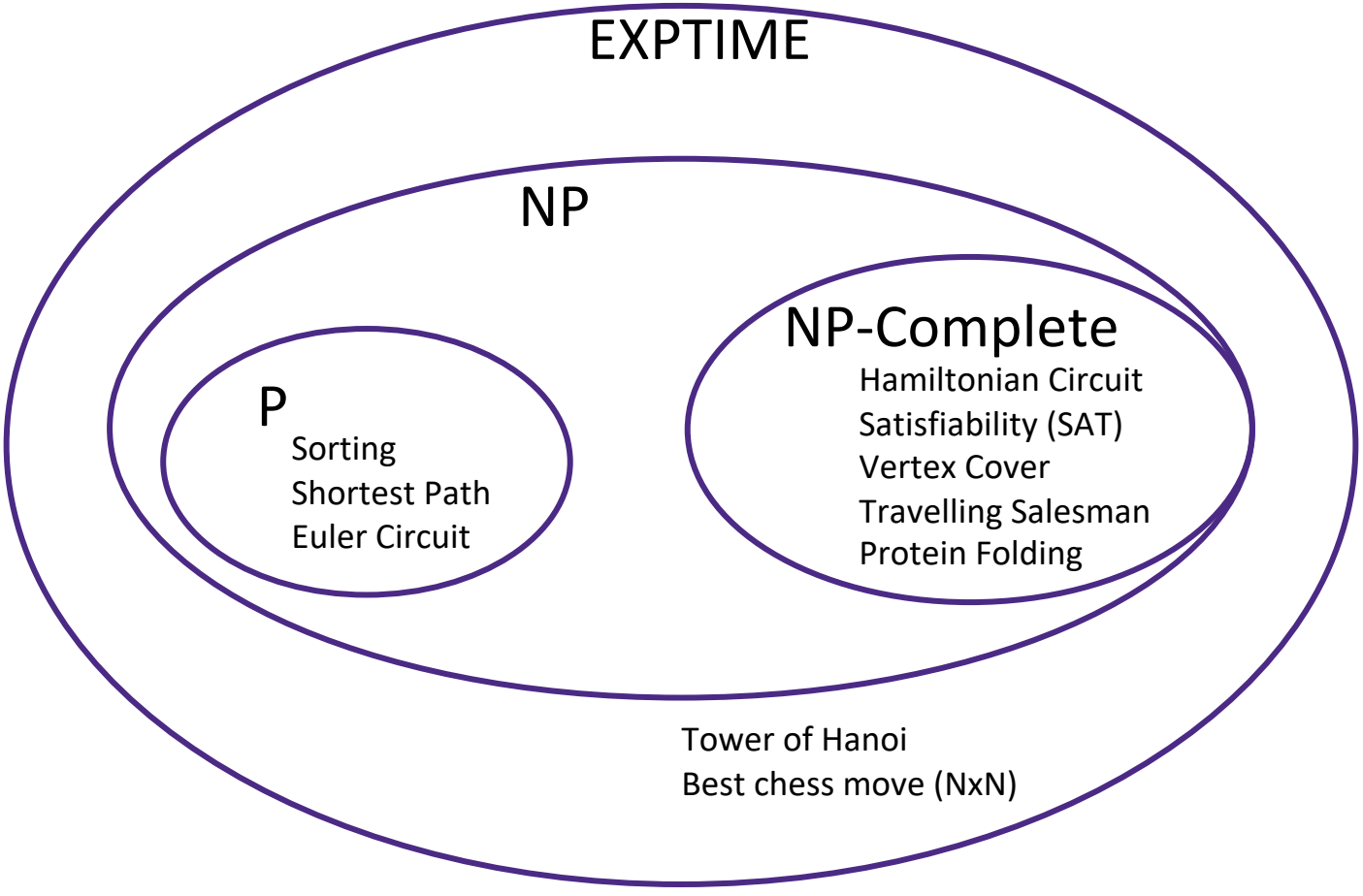
What *do* we know??

- Approximation Algorithm:
  - Can we get an efficient algorithm that guarantees something *close* to optimal? (e.g. Answer is guaranteed to be within 1.5x of Optimal, but solved in polynomial time).
- Restrictions:
  - Many hard problems are easy for restricted inputs (e.g. graph is always a tree, degree of vertices is always 3 or less).
- Heuristics:
  - Can we get something that seems to work well (good approximation/fast enough) *most* of the time? (e.g. In practice, n is small-ish)

Great Quick Reference

- *Computers and Intractability: A Guide to the Theory of NP-Completeness*, by Michael S. Garey and David S. Johnson




EXPTIME
NP
NP-Complete
Hamiltonian Circuit
Satisfiability (SAT)
Vertex Cover
Travelling Salesman
Protein Folding
P
Sorting
Shortest Path
Euler Circuit
Tower of Hanoi
Best chess move (NxN)
Halting

Summary

- P and NP are defined in terms of different attributes (i.e., solvability vs verifiability), and we suspect that this means P is a proper subset of NP

- Thanks to reductions, NP-Complete problems are the "hardest" in NP. They are also "characteristic" of NP problems
  - More on this, plus examples on reductions in 421!

- There are multiple complexity classes outside of NP
  - More on this in 431!