P vs NP

CSE 332 Spring 2021

Instructor: Hannah C. Tang

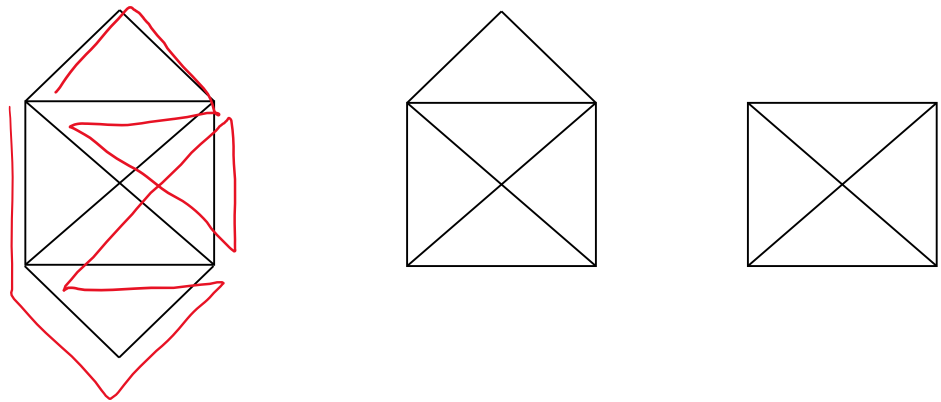
Teaching Assistants:

Aayushi Modi	Khushi Chaudhari	Patrick Murphy
Aashna Sheth	Kris Wong	Richard Jiang
Frederick Huyan	Logan Milandin	Winston Jodjana
Hamsa Shankar	Nachiket Karmarkar	

gradescope

gradescope.com/courses/256241

- Which of these can you draw (ie, trace all edges) without lifting your pencil, *drawing each line only once*?
  - Can you start and end at the same point?



- Enumerate 1-2 algorithms with the following worst-case runtimes:
  - O(log n)
  - O(n)
  - $O(n^2)$ or $O(V^2)$ or $O(E^2)$
  - O(V + E)

Announcements

- Please fill out course evals!
- Please nominate your TAs for the Bob Bandes Award!!
  - They deserve it!
- Quiz review and section showdown tomorrow; winners crowned on Friday

Lecture Outline

- Circuits
  - **Euler Circuit**
  - Hamiltonian Circuit

- Complexity classes
  - P and non-P
  - A Whirlwind Tour of non-P Problems
  - NP

Setting Up A Prank

- Your friend is organizing a tour of local farmland and wants donors to drive over every road in the Snoqualmie River Valley
- Driving over the roads costs money (fuel), and there are a lot of roads
- She wants you to figure out how to drive over *each road exactly once*, returning to your starting point
- *(note: this didn't actually happen)*

Euler Circuits

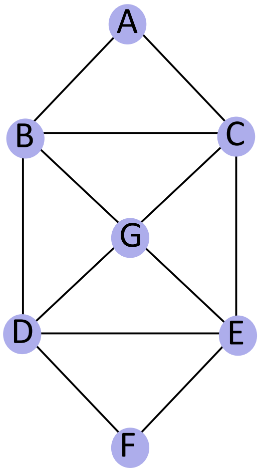
- ***Euler Circuit***: a path through a graph that visits each edge exactly once, and starts and ends at the same vertex

- Named after Leonhard Euler (1707-1783), who cracked this problem and founded graph theory in 1736
  - This problem is also known as "the Seven Bridges of Königsberg"

- An Euler circuit exists iff
  - The graph is connected and
  - Each vertex has even degree (= # of edges on the vertex)

gradescope

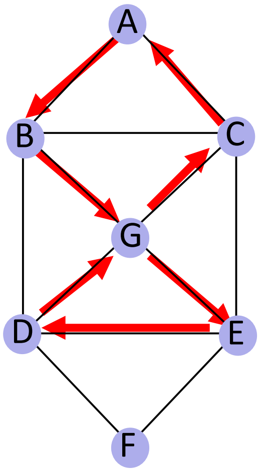
gradescope.com/courses/256241



Euler(A):

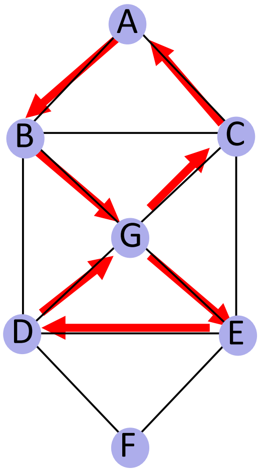
- Find an Euler circuit starting at A

Euler Circuit: Example

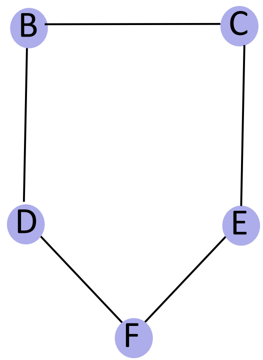


Euler(A):
A B G E D G C A

Euler Circuit: Example

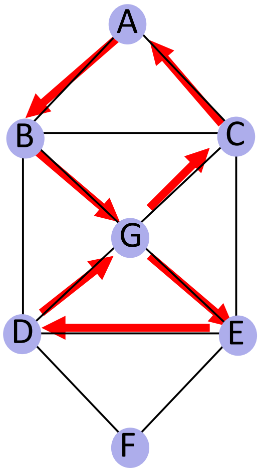


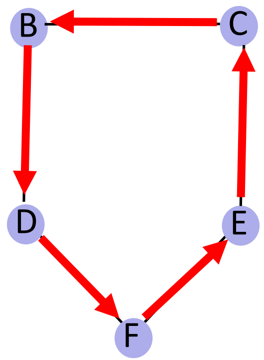
Euler(A):
A B G E D G C A



Euler(B):

Euler Circuit: Example



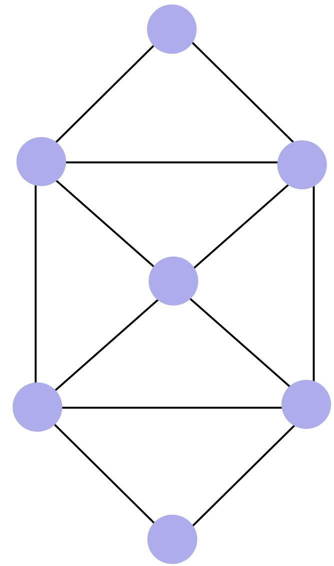


Euler(A):
A **B** G E D G C A

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

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

Euler Circuit: Algorithm



- Given a connected undirected graph G = (V,E)
- Can check if a circuit exists: O(V)
  - Do all vertices have even degree?
- Can find a circuit: O(V+E)
  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: O(E)
  - Traverse path, marking visited edges
  - Return true if all edges are marked, and $v_0 == v_n$

Lecture Outline

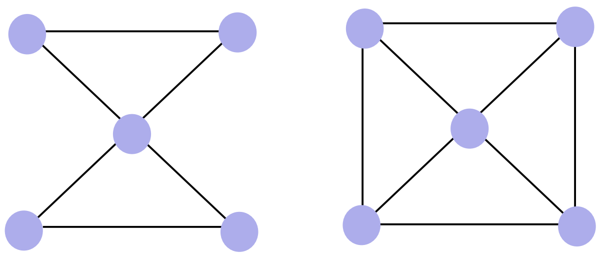
- Circuits
  - Euler Circuit
  - **Hamiltonian Circuit**
- Complexity classes
  - P and non-P
  - A Whirlwind Tour of non-P Problems
  - NP

The Actual Prank

- Instead of a farmland tour, she wanted a farm tour
- Now you need to figure out how to drive to *each farm exactly once*, returning in the first farm at the end
- *(note: this actually DID happen. I still get razzed about it)*

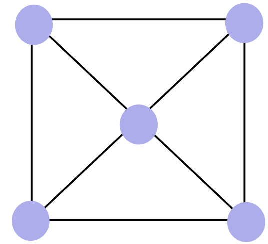
Hamiltonian Circuits

- ***Euler circuit***: a cycle that goes through each *edge* exactly once
- ***Hamiltonian circuit***: a cycle that goes through each *vertex* exactly once
- Does the first graph have:
  - An Euler circuit? Y
  - A Hamiltonian circuit? N
- Does the second graph have:
  - An Euler circuit? N
  - A Hamiltonian circuit? Y



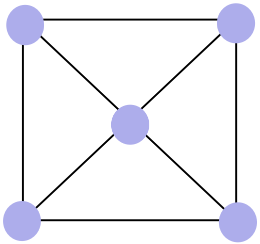
Which problem sounds harder?

Hamiltonian Circuit Verification = Good News



- Given a connected unweighted undirected graph G = (V, E)
- Can verify a given path is a circuit: O(V)
  - Traverse path, marking visited *vertices*
  - Return true if all *vertices* are marked, and $v_0 == v_n$

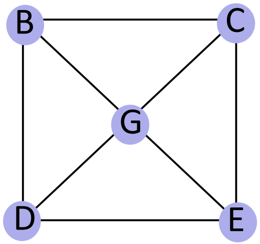
Hamiltonian Circuit Algorithm = Bad News



- Algorithm:
  - Enumerate all *paths*, check if one of them is a circuit
    - Can use your favorite graph search algorithm to enumerate paths
  - This is known as an exhaustive search ("brute force") algorithm

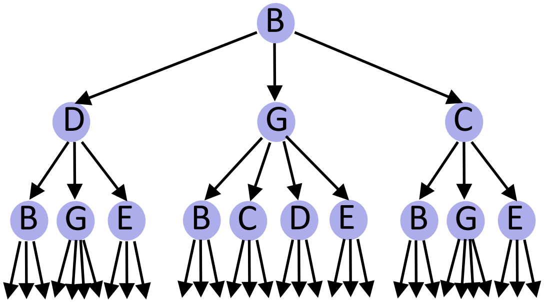
- Can find a circuit: $O(V)$
  - Enumerate all *paths*, check if one of them is a circuit

Exhaustive Search: Analysis (1 of 2)



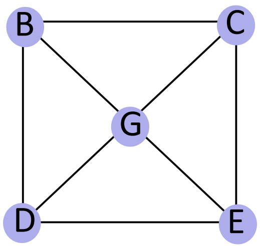
- Worst case needs to enumerate all paths
  - ***How many paths are there??***

- 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



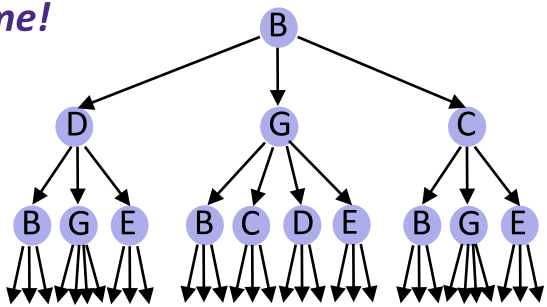
Search tree of paths starting at B

Exhaustive Search: Analysis (2 of 2)



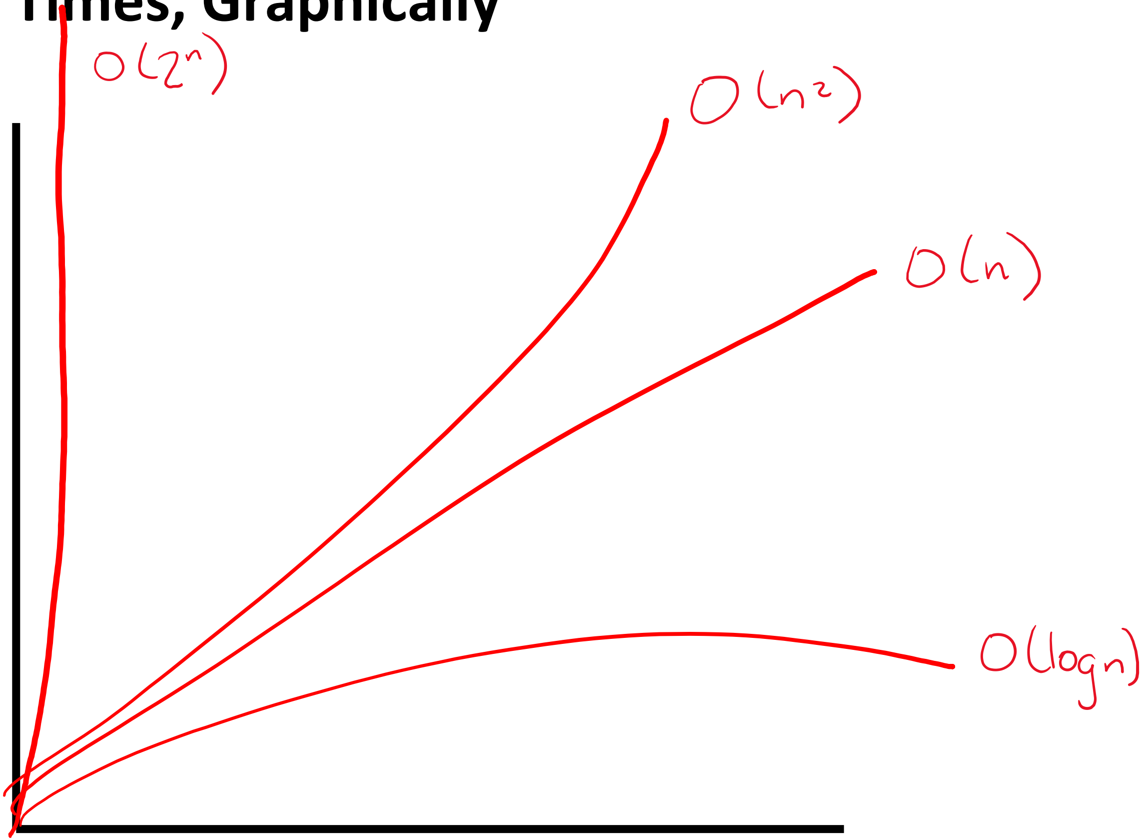
$b = \frac{16}{5} \approx 3.2$

- Let *b* be the *average* branching factor of each node in this graph
  - |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



Demo: https://www.desmos.com/calculator/diufnxtyqy

Running Times, Numerically

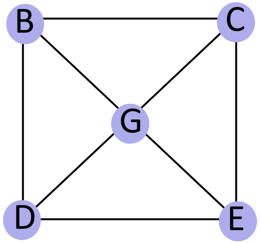
Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	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

lol

Somewhat old, from Rosen

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

- Circuits
  - Euler Circuit
  - Hamiltonian Circuit
- Complexity classes
  - **P and non-P**
  - A Whirlwind Tour of non-P Problems
  - NP

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.
- Examples of problems that are (probably) not in P:
  - Hamiltonian circuit, satisfiability (SAT), vertex cover, travelling salesman, Tower of Hanoi, etc.

P

Sorting
Shortest Path
Euler Circuit

Hamiltonian Circuit
Satisfiability (SAT)
Vertex Cover
Travelling Salesman

Tower of Hanoi
Best chess move (NxN)

Halting

Lecture Outline

- Circuits
  - Euler Circuit
  - Hamiltonian Circuit
- Complexity classes
  - P and non-P
  - **A whirlwind tour of (probably) non-P problems**
  - NP

Satisfiability

3 5

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

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

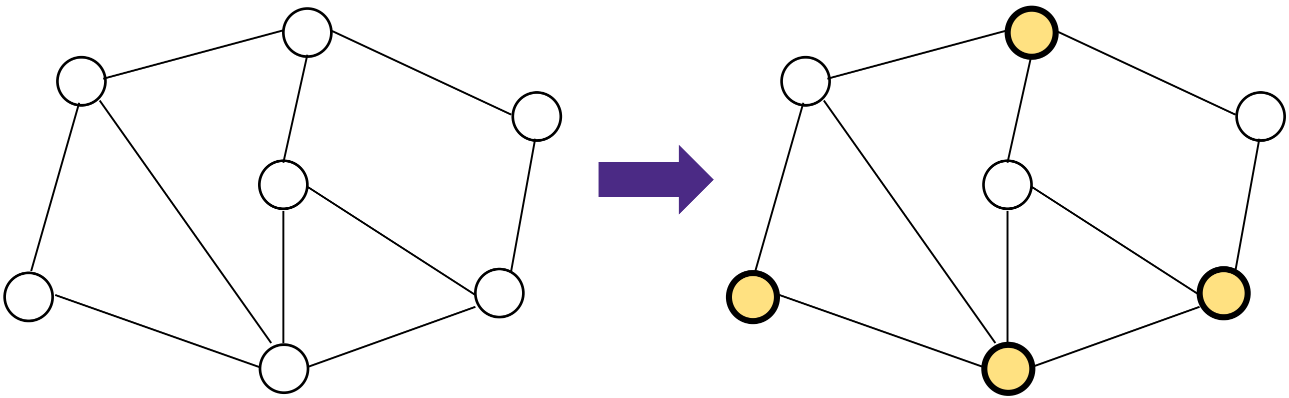
- *Algorithm*: Try every variable assignment

	Soln 1	Soln 2	…	Soln 2^n
x_1	T	F	…	F
x_2	T	T	…	F
x_3	T	T	…	F
x_4	T	T	…	F
x_5	T	T	…	F

Vertex Cover

- *Input*: A graph G = (V,E) and a number *m*

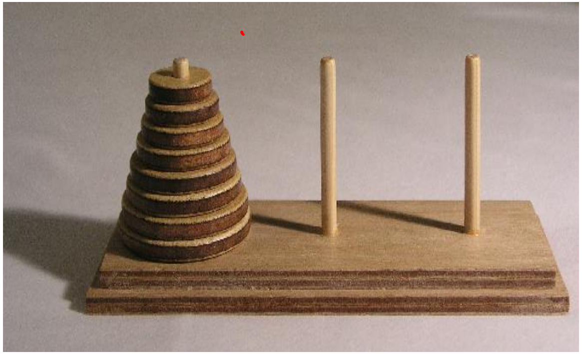
- *Output*: A subset S of V, such that:
  - For every edge (u,v) in E, at least one of u or v is in S
  - |S|=*m* (if such an S exists)



- *Algorithm*: Try every subset of vertices of size *m*

Tower of Hanoi

- *Input*: *n* disks of increasing size and 3 pegs
- *Output*: A series of moves transferring *n* disks to any other peg without placing a larger disk over a smaller one



- Algorithm:

```
while (!done):
  transferDisk(peg A, peg B)
  transferDisk(peg A, peg C)
  transferDisk(peg B, peg C)
```

Runtime: $O(2^n)$
Length of solution: $O(2^n)$

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

Lecture Outline

- Circuits
  - Euler Circuit
  - Hamiltonian Circuit
- Complexity classes
  - P and non-P
  - A whirlwind tour of (probably) non-P problems
  - **NP**

A Glimmer of Hope?

- 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*** 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*

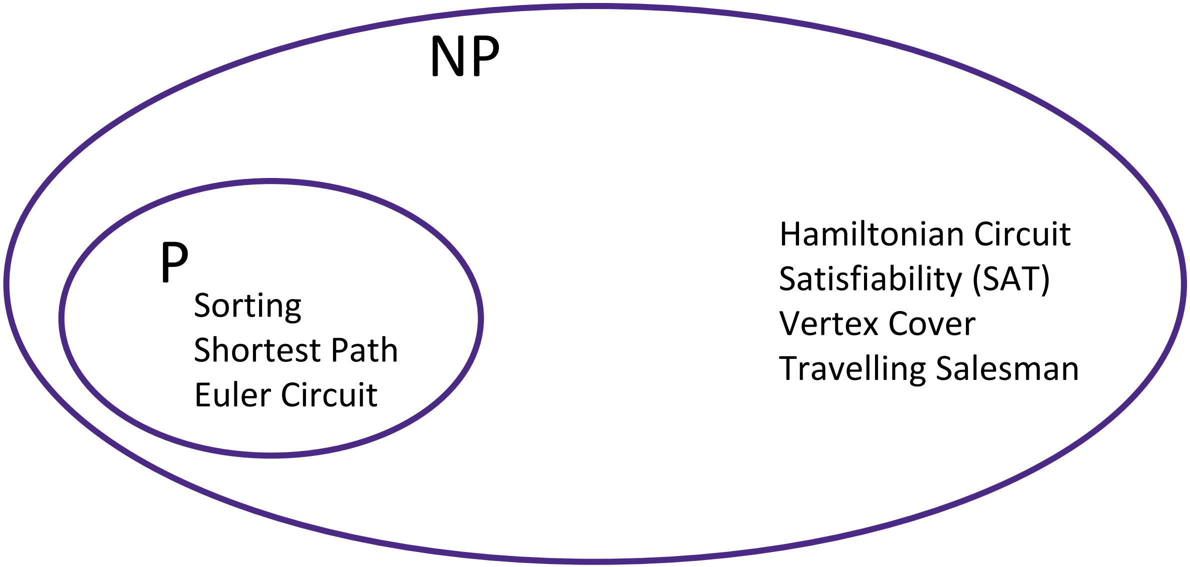
- Examples of problems in NP:
  - *Hamiltonian circuit*: Given a candidate path, can verify in O(|V|) time if it is a Hamiltonian circuit
  - *Satisfiability*: Given a candidate set of *n* values, can verify in O(*m*) time if the expression is true
  - *Vertex Cover*: Given a subset of vertices, can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



Tower of Hanoi
Best chess move (NxN)

Halting

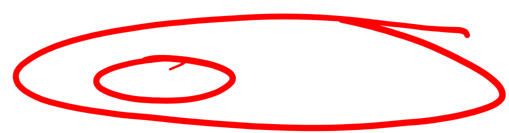
Your Chance to Win a Turing Award!

- 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 !

P=NP

P!=NP



Summary

- One small change from *edges* to *vertices* changed the Euler Circuit problem into the Hamiltonian Circuit problem
  - … and had a huge impact on the algorithm's runtime
- P is characterized by the runtime of its *solutions*
  - Must be able to solve in polynomial time
- NP is characterized by the runtime of its *verifications*
  - No constraints on time to solve
  - Must be able to verify in polynomial time