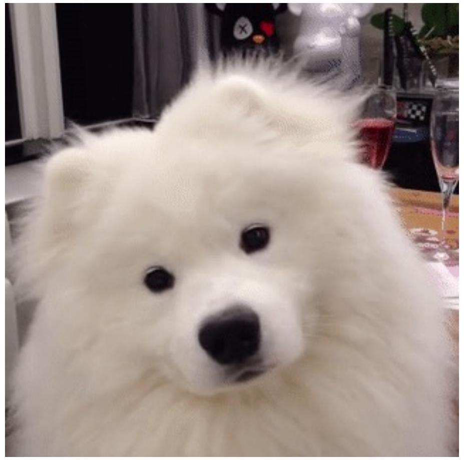
CSE 332: Data Structures & Parallelism
Lecture 22: P, NP, NP-Complete



Arthur Liu

Summer 2022

Announcements

- Reminder your final is on **two days,** Section 10/18, Lecture 10/19
    - Make sure to be in your correct quiz section on Thursday for pt1. of the exam! We will take attendance, so bring student ID to section
- Final Review Session: MOR 220 Wed 10/17 from 3:00-4:00pm
- Exam Topics and Practice Exams on the website!
    - Make sure to look at some past finals to practice!

Class Survey!

- You should have received an email for a survey for this class!
    - It closes this Friday!
- I will give out 1 extra credit point to everyone who fills it out
    - It is anonymous, I will know if you filled it out but not what you said

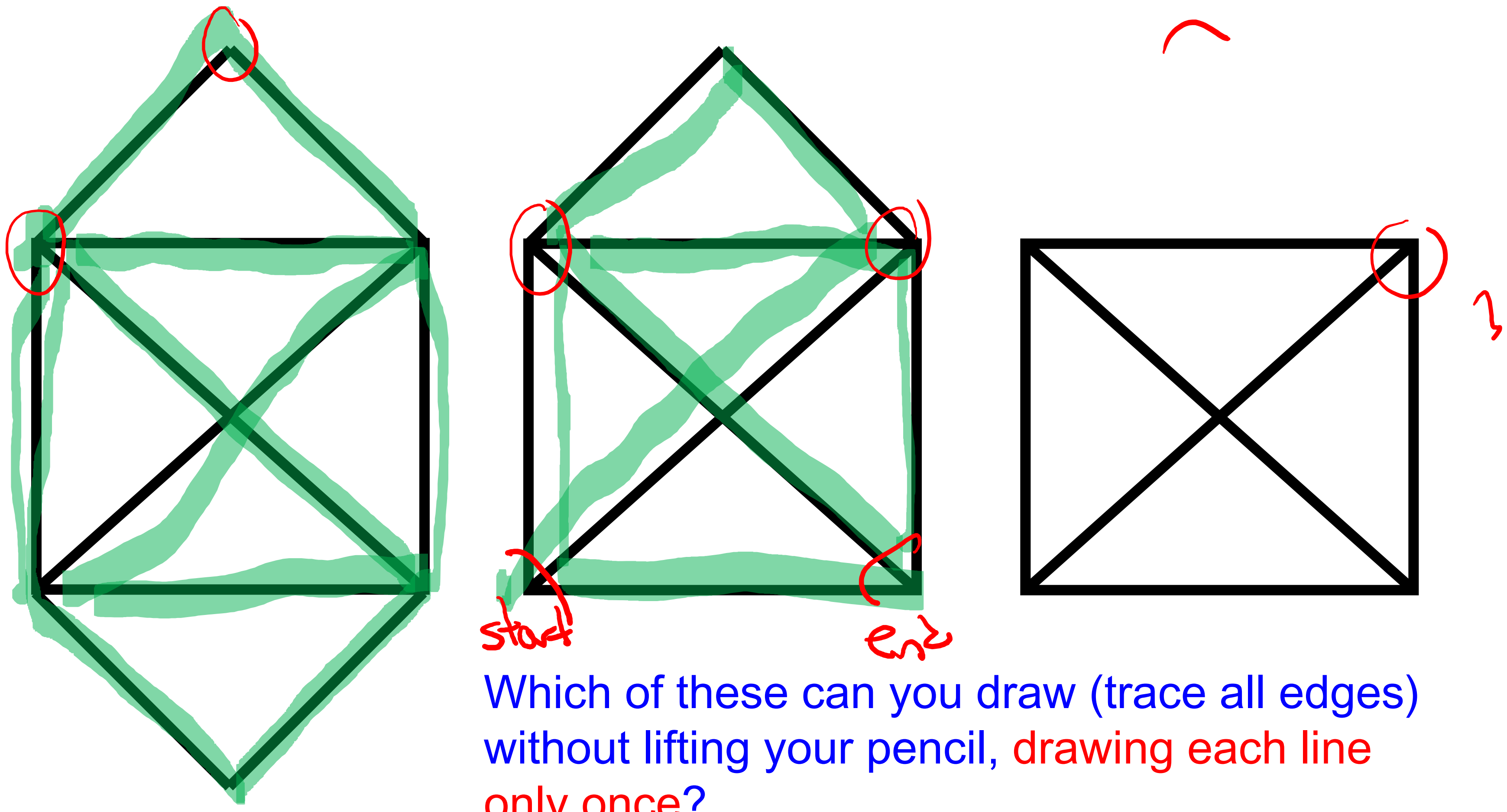
Outline

- A Few Problems:
  - Euler Circuits
  - Hamiltonian Circuits
- P and NP
- NP-Complete
- What now?

Try it

degree

even

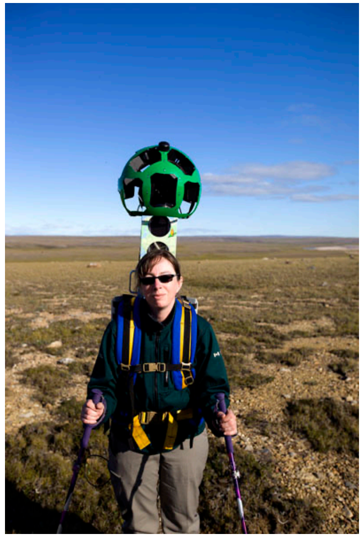


Which of these can you draw (trace all edges) without lifting your pencil, drawing each line only once?

Can you start and end at the same point?

Your First Task

- 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.



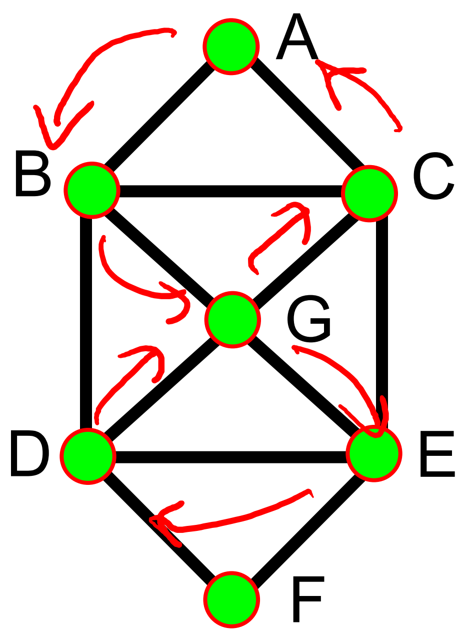
Euler Circuits

- 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

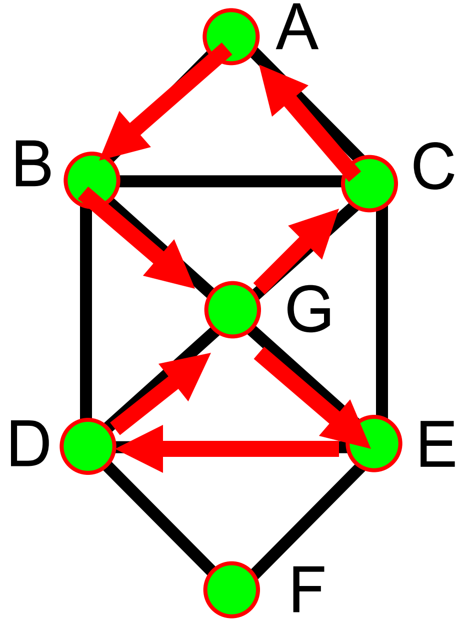
- A Euler circuit exists *iff*
  - the graph is connected and
  - each vertex has even degree (= # of edges on the vertex)

Euler Circuit Example



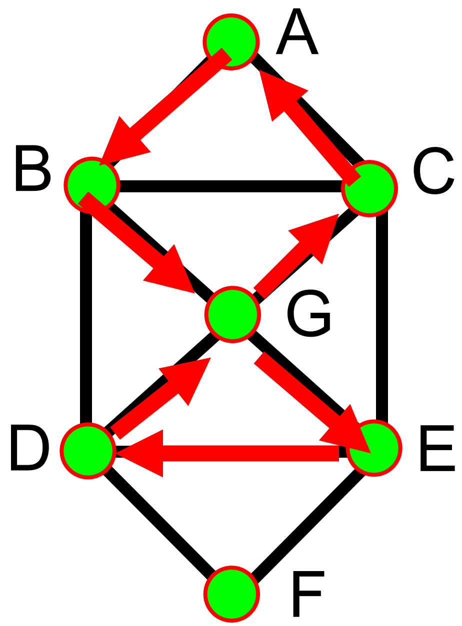
Euler(A) :

Euler Circuit Example

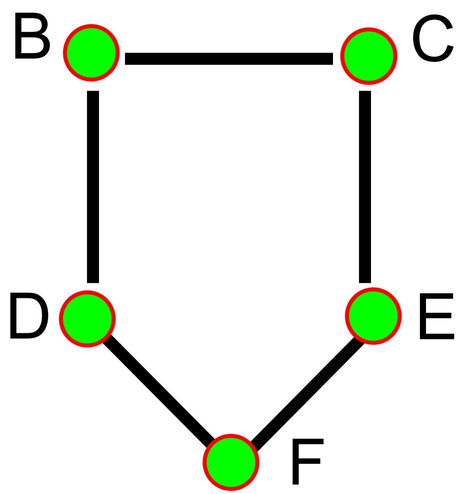


Euler(A) :
A B G E D G C A

Euler Circuit Example

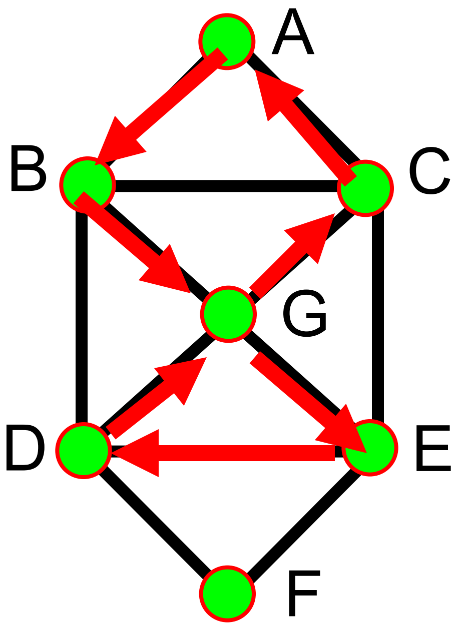


Euler(A) :
A B G E D G C A

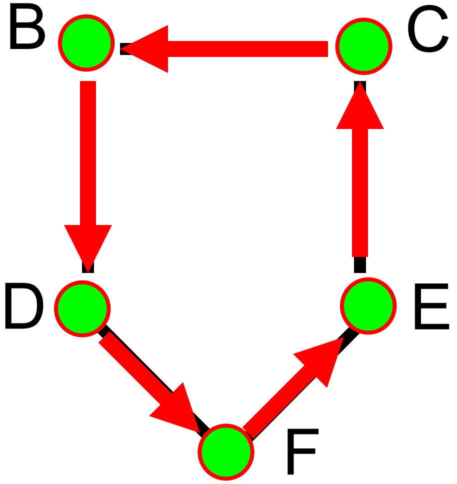


Euler(B)

Euler Circuit Example

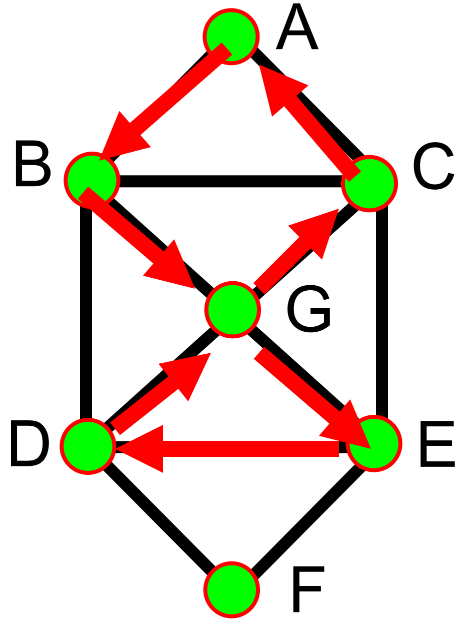


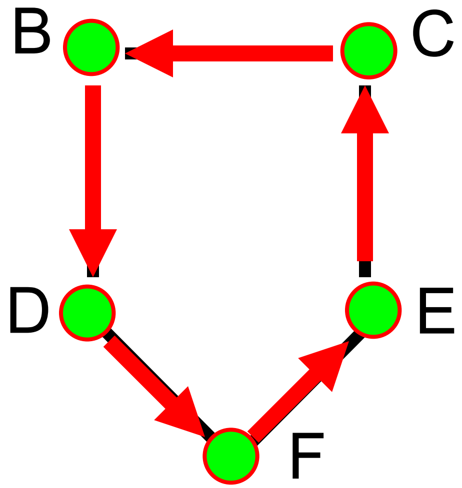
Euler(A) :
A B G E D G C A



Euler(B):
B D F E C B

Euler Circuit Example





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

Euler(B):
B D F E C B

Splice

A B D F E C B G E D G C A

The Road Inspector: Finding Euler Circuits

Given a connected, undirected graph G = (V,E), find an Euler circuit in G

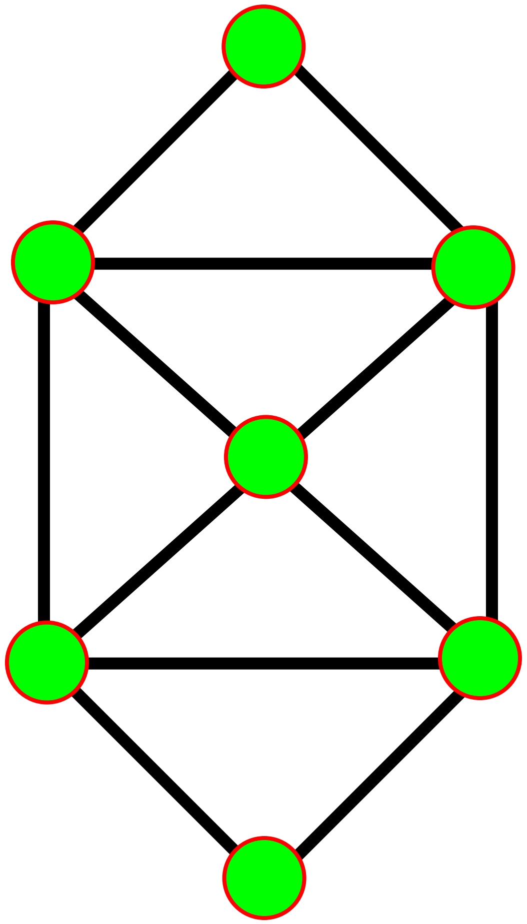
Can check if one exists:

- Check if all vertices have even degree → O(V+E)

Basic Euler Circuit Algorithm:

1. Do an edge walk from a start vertex until you are back to the start vertex.
   - You never get stuck because of the even degree property.
2. "Remove" the walk, leaving several components each with the even degree property.
   - Recursively find Euler circuits for these.
3. Splice all these circuits into a Euler circuit

O(E)



Running time?

The Road Inspector: Finding Euler Circuits

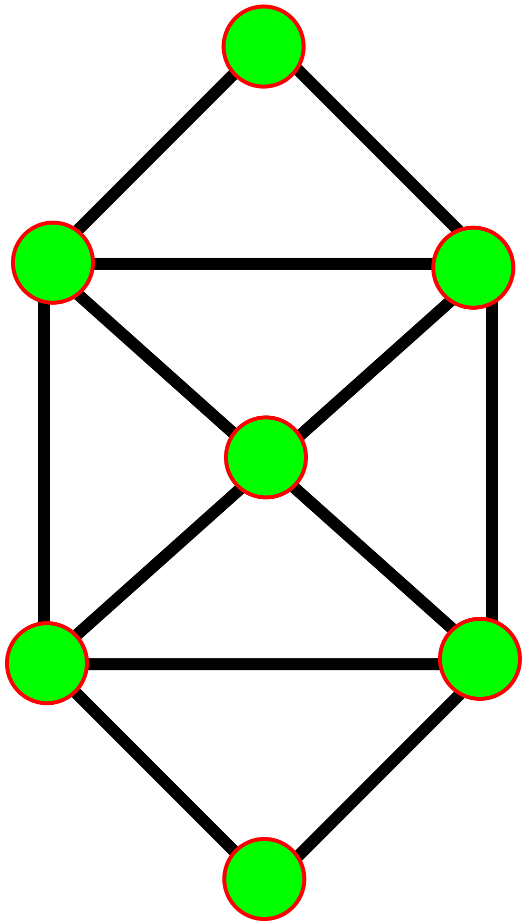
Given a connected, undirected graph G = (V,E), find an Euler circuit in G

Can check if one exists: (in O(|V|+|E|))

- Check if all vertices have even degree

Basic Euler Circuit Algorithm:

1. Do an edge walk from a start vertex until you are back to the start vertex.
   - You never get stuck because of the even degree property.
2. "Remove" the walk, leaving several components each with the even degree property.
   - Recursively find Euler circuits for these.
3. Splice all these circuits into a Euler circuit



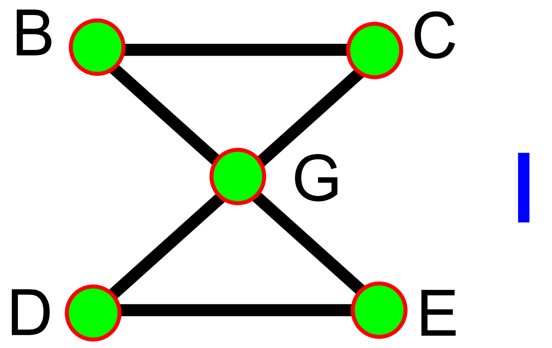
Running time? O(|V|+|E|)

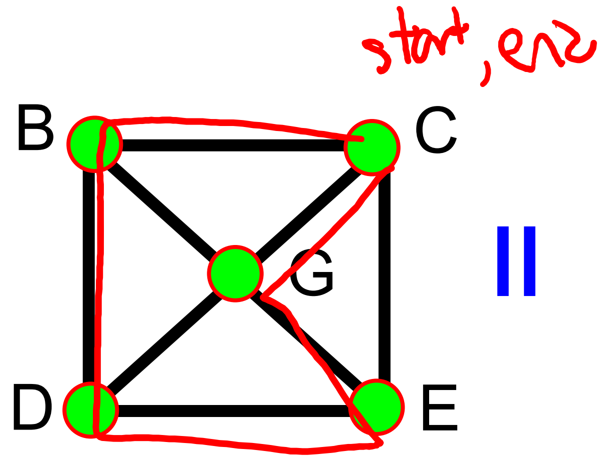
Your Second Task

- Your boss is pleased…and assigns you a new task.
- Your company has to send someone by car to a set of cities.
- The primary cost is the exorbitant toll going into each city.
- Your boss wants you to figure out *how to drive to each city exactly once, returning in the end to the city of origin*.

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 graph I have:
  - An Euler circuit? Yes
  - A Hamiltonian circuit? No
- Does graph II have:
  - An Euler circuit? No
  - A Hamiltonian circuit? Yes
- Which problem sounds harder?





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

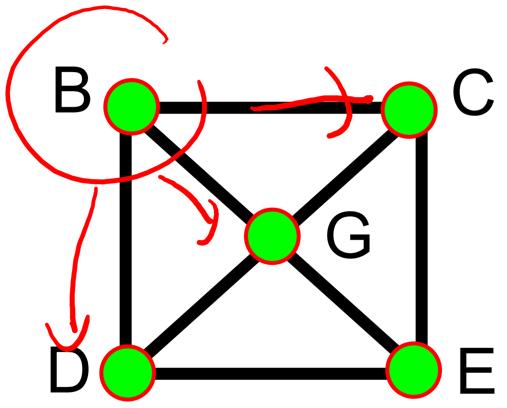
- Worst case: need to search all paths
  - How many paths??

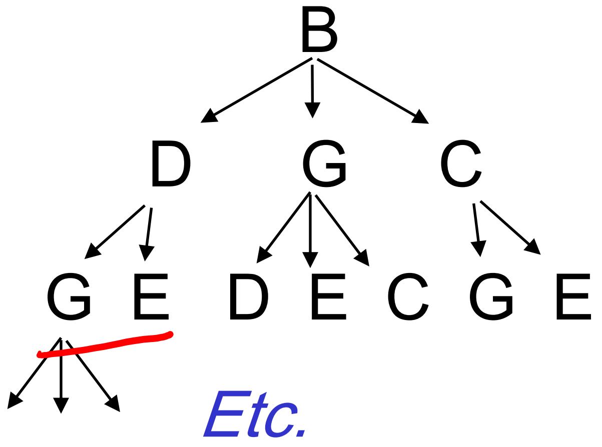
Analysis of Exhaustive Search Algorithm

Worst case: need to search all paths

- How many paths?

Can depict these paths as a *search tree:*





Search tree of paths from B

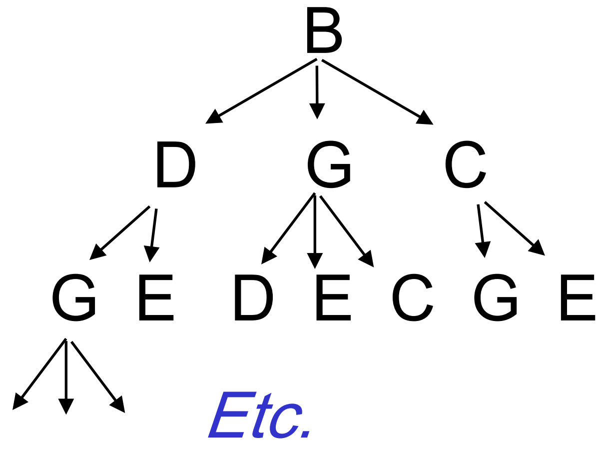
Analysis of Exhaustive Search Algorithm

Let the *average* branching factor of each node in this tree be b

|V| vertices, each with ≈ b branches

Total number of paths ≈ b·b·b … ·b (V times)

Worst case → $O(b^V)$ → $O(2^N)$



Search tree of paths from B

Analysis of Exhaustive Search Algorithm

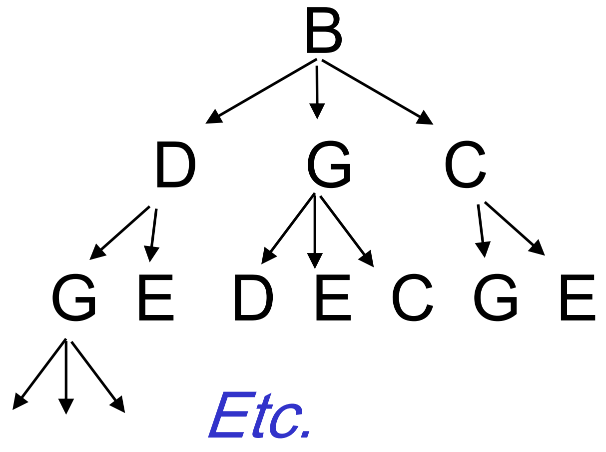
Let the *average* branching factor of each node in this tree be b

|V| vertices, each with ≈ b branches

Total number of paths ≈ b·b·b … ·b

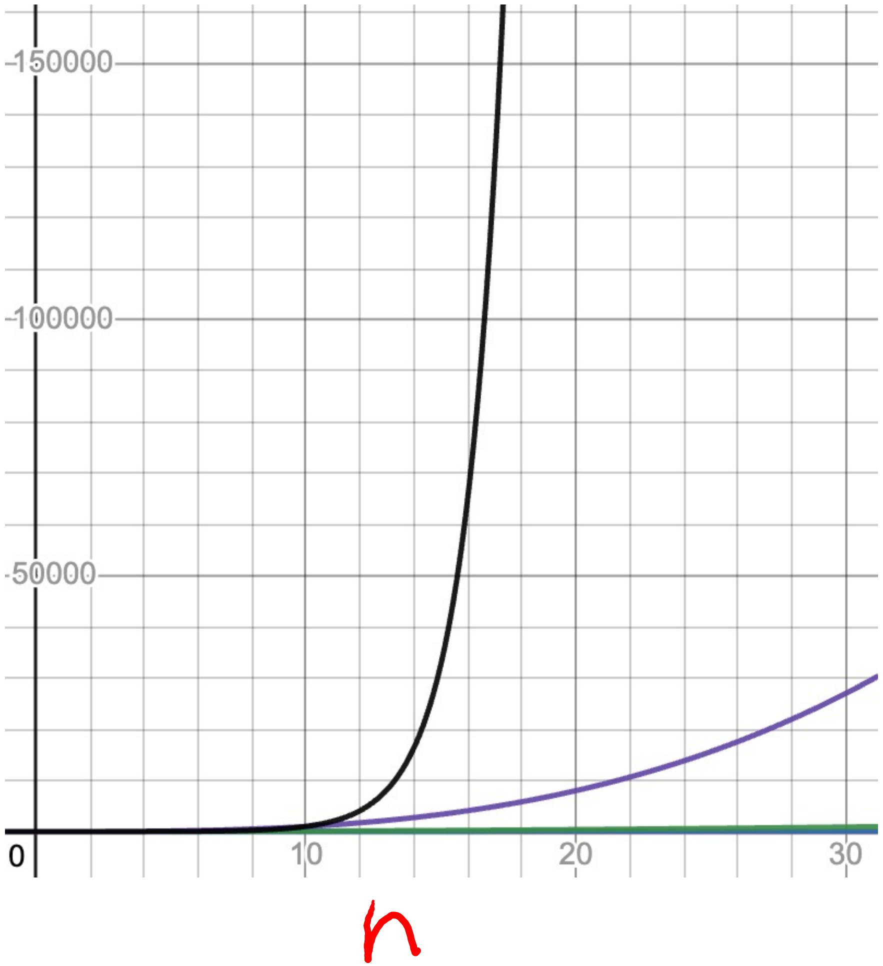
$O(b^{|V|})$

Worst case → Exponential time!



Search tree of paths from B

Running Times



https://www.desmos.com/calculator/pwxhtawjnx

More Running Times

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

Polynomial vs. Exponential Time

Euler easy
edges

- All of the algorithms we have discussed in this class have been **polynomial time** algorithms:
  - Examples: O(log N), O(N), O(N log N), $O(N^2)$
  - Algorithms whose running time is $O(N^k)$ for some k > 0

- **Exponential time** b^N is asymptotically worse than any polynomial function N^k for any k

Very
Hard

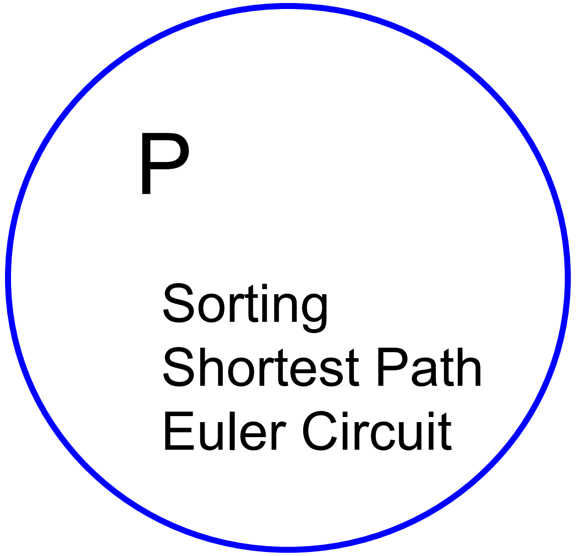
vertices

The Complexity Class P

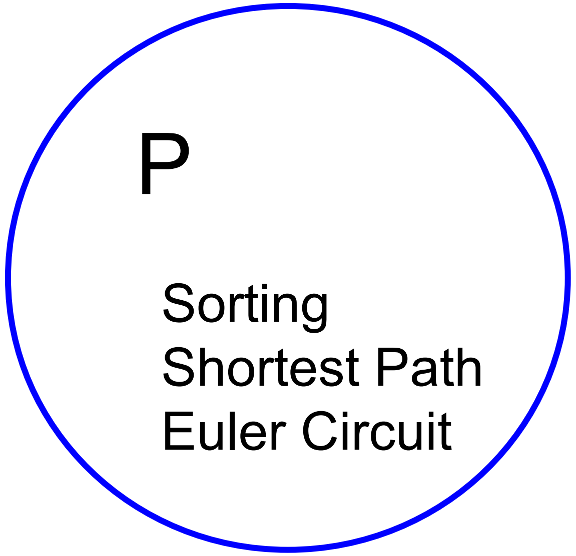
Definition: P is the set of all problems that can be solved in *polynomial worst-case time*

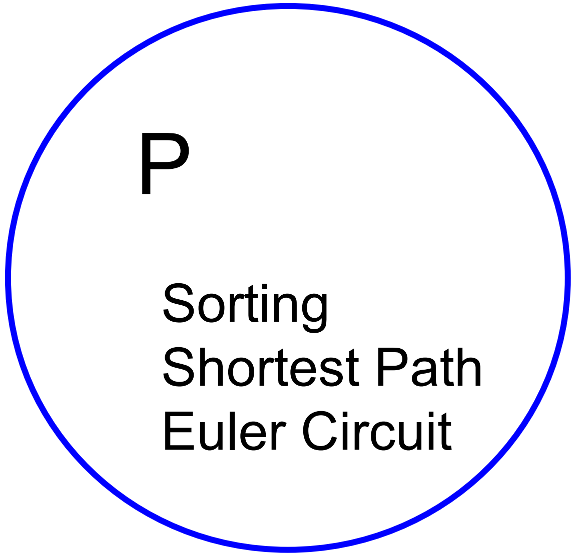
All *problems* that have some *algorithm* whose running time is $O(N^k)$ for some k

Examples of problems in P:
sorting, shortest path, Euler circuit, *etc*.


P
Sorting
Shortest Path
Euler Circuit

Hamiltonian Circuit





Hamiltonian Circuit
Satisfiability (SAT)
Vertex Cover
Travelling Salesman

Satisfiability

Input: a logic formula of size **m** containing **n** variables

Output: An assignment of Boolean values to the variables in the formula such that the formula is true

Algorithm: Try every variable assignment

$$(\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee \neg x_5)$$

$x_1 = T$

$x_2 = T$

$x_4 = T$

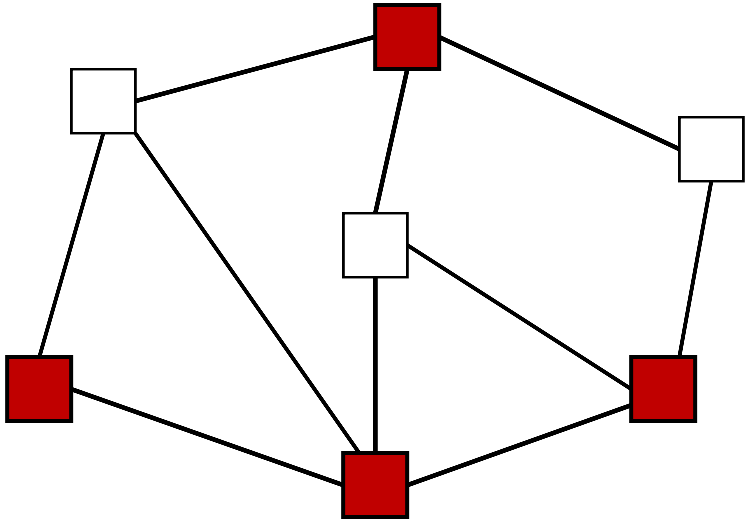
$x_1 = T$

Vertex Cover

Input: A graph (**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** and **|S|=m** (if such an **S** exists)

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



Traveling Salesman

Input: A *complete* *weighted* graph (**V,E**) and a number **m**

Output: A circuit that visits each vertex exactly once and has total cost < **m** if one exists

Algorithm: Try every path, stop if find cheap enough one

A Glimmer of Hope

If given a candidate solution to a problem, we can <u>check if that solution is correct in polynomial-time</u>, then **maybe** a polynomial-time solution exists?

Can we do this with Hamiltonian Circuit?

- Given a candidate path, is it a Hamiltonian Circuit?

A Glimmer of Hope

If given a candidate solution to a problem, we can **check if that solution is correct in polynomial-time**, then **maybe** a polynomial-time solution exists?

Can we do this with Hamiltonian Circuit?

Given a candidate path, is it a Hamiltonian Circuit?

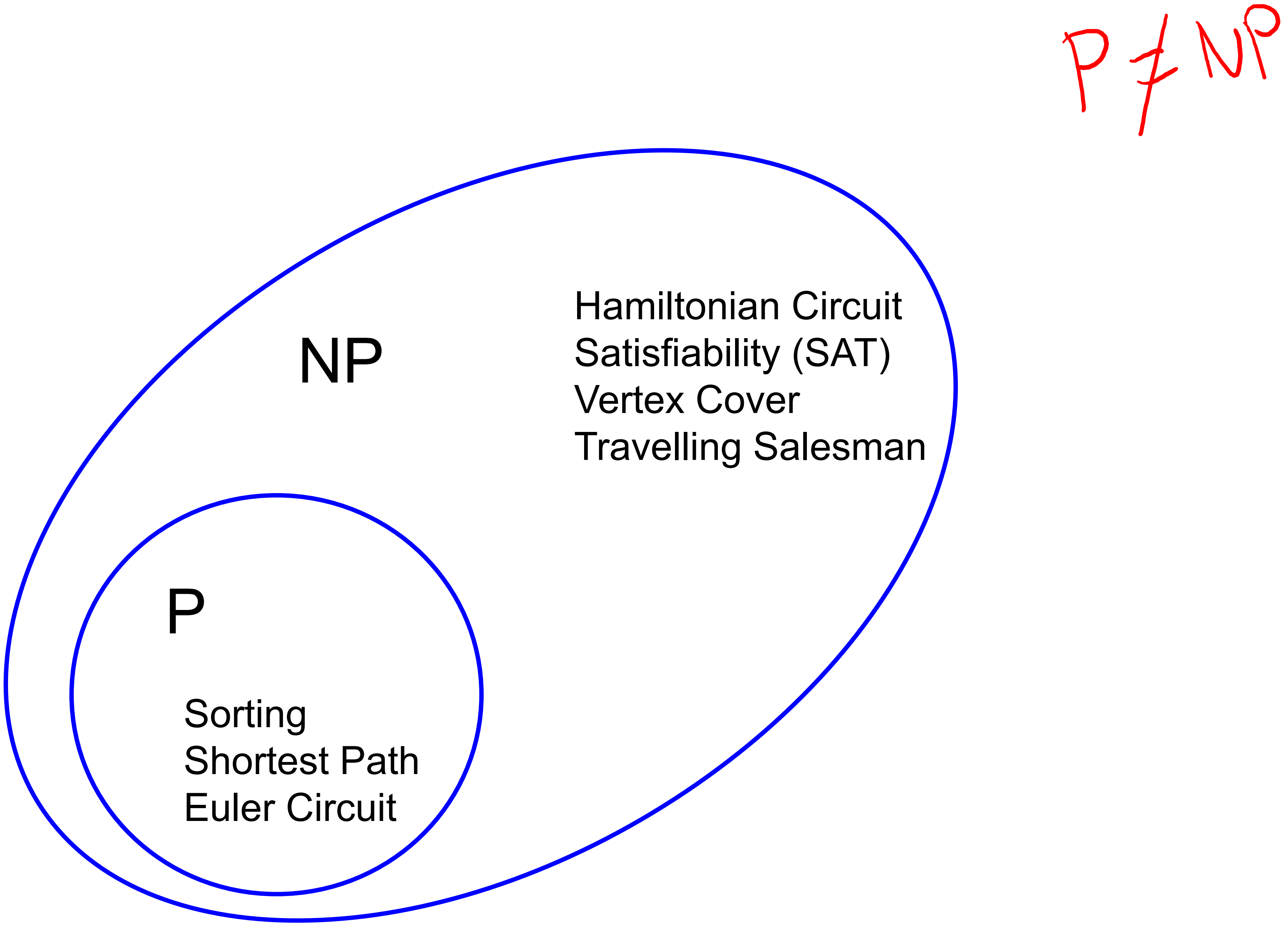
just check if all vertices are visited exactly once in the candidate path

The Complexity Class NP

Definition: NP is the set of all problems for which a given *candidate solution* can be *tested* in polynomial time

Examples of problems in NP:

- *Hamiltonian circuit:* Given a candidate path, can test in linear time if it is a Hamiltonian circuit
- *Vertex Cover:* Given a subset of vertices, do they cover all edges?
- *All problems that are in P (why?)*


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

Why do we call it "NP"?

- NP stands for *Nondeterministic Polynomial time*
  - Why "nondeterministic"? Corresponds to algorithms that can guess a solution (if it exists), the solution is then verified to be correct in polynomial time
  - Can also think of as allowing 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

It does NOT stand for "non-polynomial"

Reductions

- Let's say we want to make some claim about NP problems, we would want to pick the "most difficult" NP problem as our representative.
- What does it mean for one problem to be harder than another?

Polynomial Time Reducible
We say A reduces to B in polynomial time, if there is an algorithm that, using a (hypothetical) polynomial-time algorithm for B, solves problem A in polynomial-time.

Another way of thinking about it

- Problem A ***reduces*** to Problem B
- Problem A *"can be converted"* to Problem B
  - Problem B is the "broader, harder" problem.
  - If we can solve problem B, we can solve problem A.

Problem A: I want to solve a math equation with only addition

reduce →

Problem B: I want to solve a math equation with any operators

WolframAlpha

NP-complete

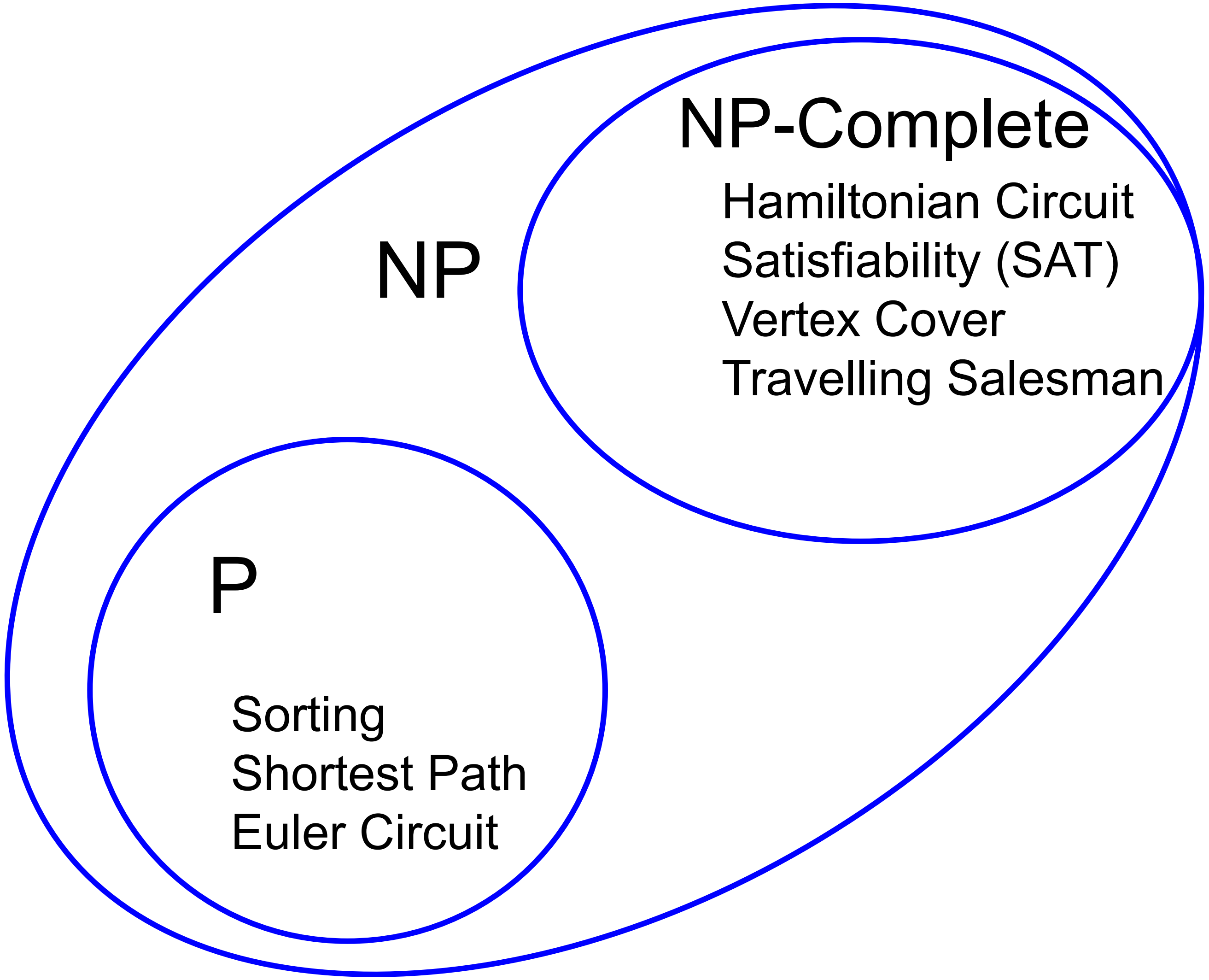
- Let’s say we want to make some claim about NP problems, we would want to pick the “most difficult” NP problem as our representative.
- What does it mean for one problem to be harder than another?

NP-complete
a problem B is NP-complete if B is in NP and for all problems A in NP, A reduces to B 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…

…and *all* NP problems can be solved in polynomial time

What The World Looks Like (We Think)

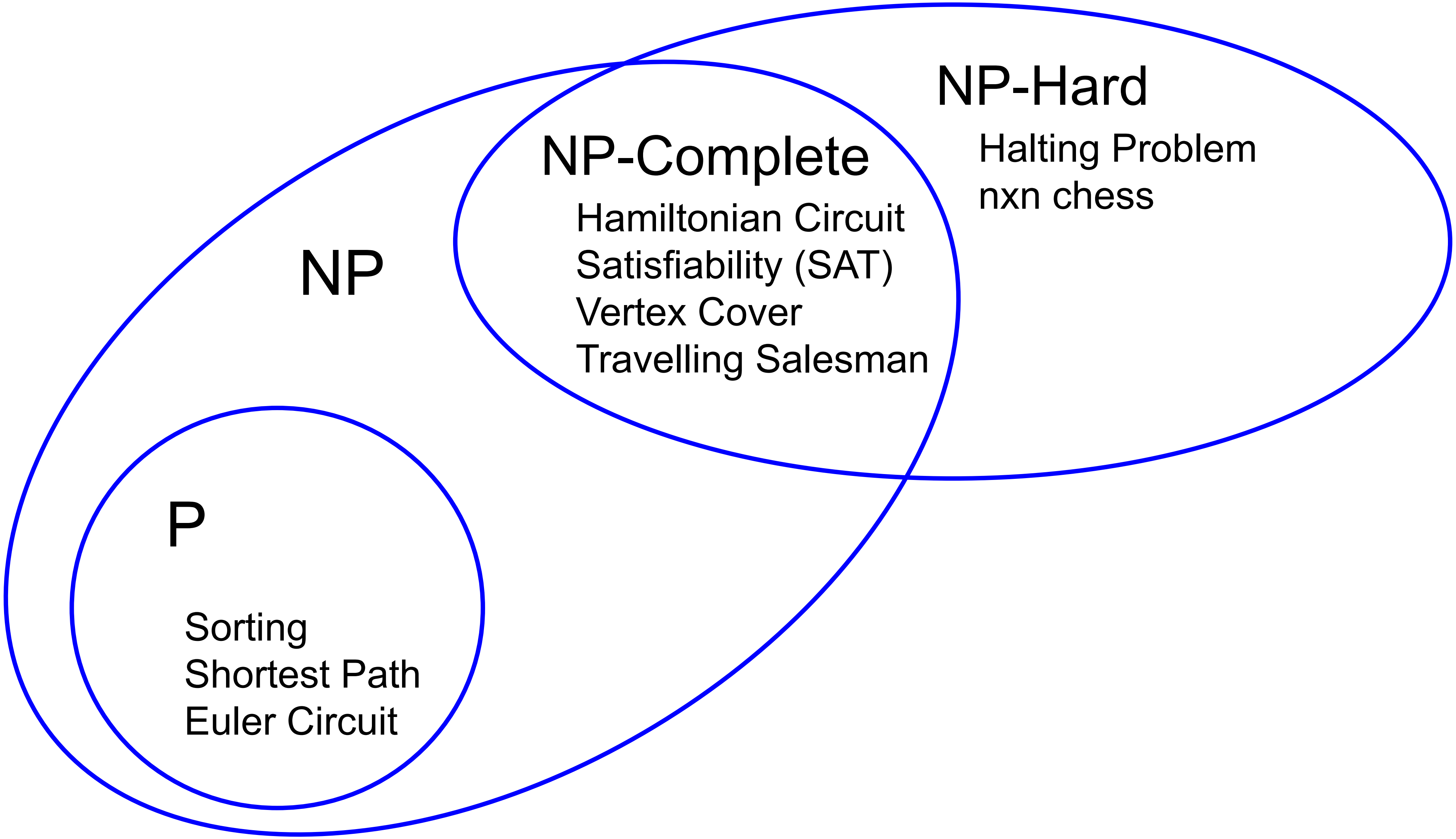


One more class – NP-Hard

NP-complete
a problem B is NP-complete if B is in NP and for all problems A in NP, A reduces to B in polynomial time.

NP-Hard
a problem B is NP-hard if for all problems A in NP, A reduces to B in polynomial time.

What The World Looks Like (We Think)



Your Chance to Win a Turing Award!

P = NP

It is generally believed that P ≠ NP,

i.e. prove 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 !

Saving Your Job

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.
- Your options:
  - Keep working
  - Come up with an alternative plan...

In general, what to do with a Hard Problem

- Your problem seems really hard.
- If you can transform a known NP-complete problem into the one you're trying to solve, then you can stop working on your problem!

Your Third Task

- Your boss buys your story that others couldn't solve the last problem.
- Again, your company has to send someone by car to a set of cities. There is a road between every pair of cities.
- The primary cost is distance traveled (which translates to fuel costs).
- Your boss wants you to figure out *how to drive to each city exactly once*, then return to the first city, while *staying within a fixed mileage budget k*.

Travelling Salesman Problem (TSP)

- Your third task is basically TSP:
  - Given <u>complete</u> weighted graph G, integer k.
  - Is there a cycle that visits all vertices with cost $<=$ k?
- One of the canonical problems.

- Note difference from Hamiltonian cycle:
  - graph is complete
  - we care about weight.

Transforming Hamiltonian Cycle to TSP

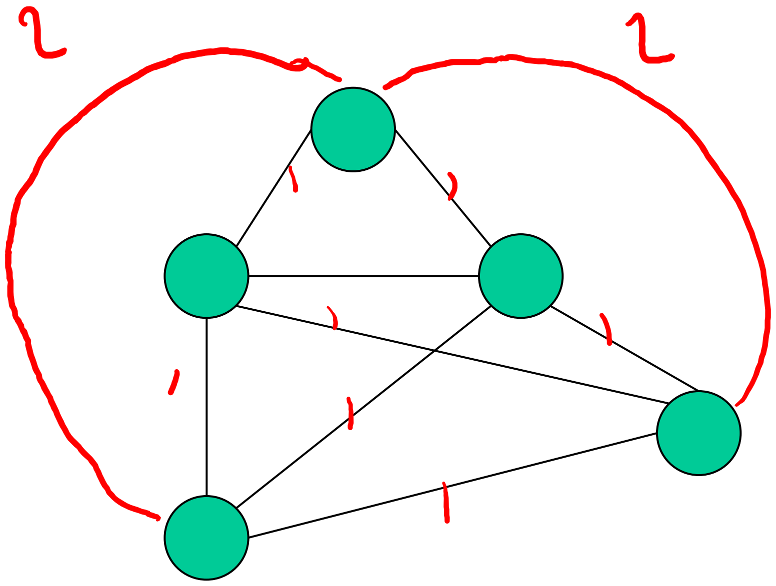
- We can “reduce” Hamiltonian Cycle to TSP.
- Given graph G=(V, E):
  - Assign weight of 1 to each edge
  - Augment the graph with edges until it is a complete graph G’=(V, E’)
  - Assign weights of 2 to the new edges
  - Let k = |V|.

Notes:

- The transformation must take polynomial time
- You reduce the known NP-complete problem into your problem (not the other way around)
- In this case we are assuming Hamiltonian Cycle is our known NP-complete problem (in reality, both are known NP-complete)

Example

$O(V^2)$
$+$
$O(E)$

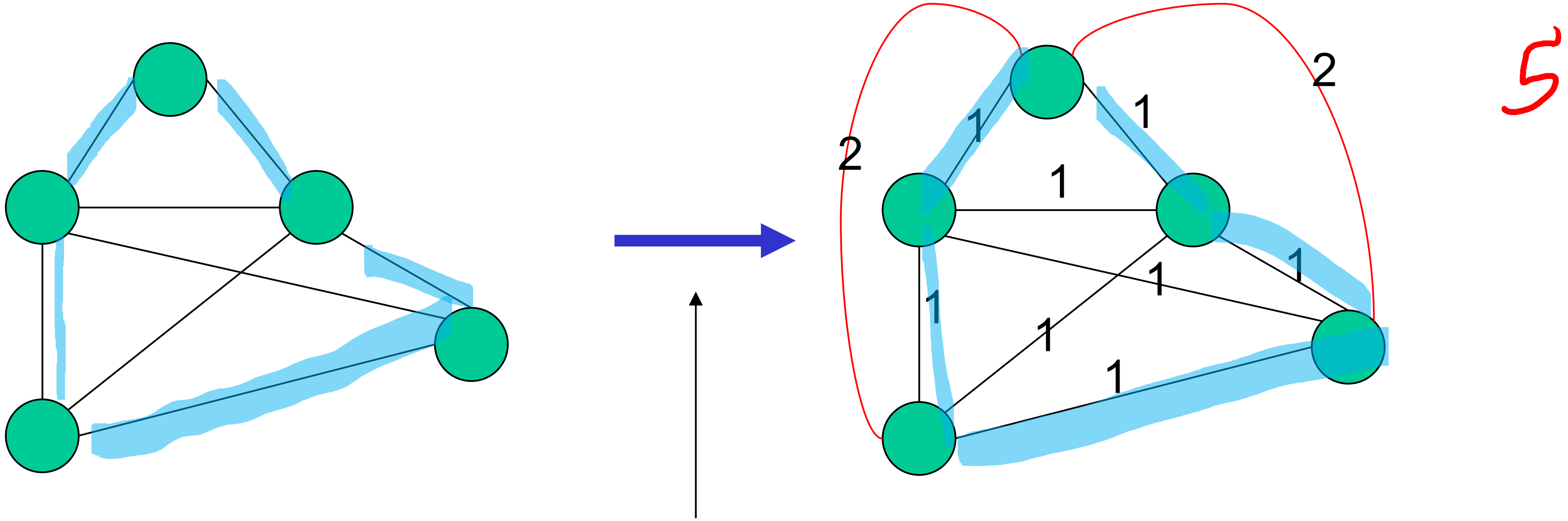


G

Input to Hamiltonian Circuit Problem

Example

budget = 5



G

Polynomial time transformation

G'

Input to Hamiltonian Circuit Problem

Hard

Input to Traveling Salesman Problem

Polynomial-time transformation

- G' has a TSP tour of weight |V| iff G has a Hamiltonian Cycle.
- What was the cost of transforming HC into TSP?

- In the end, because there is a polynomial time transformation from HC to TSP, we say *TSP is "at least as hard as" Hamiltonian cycle.*

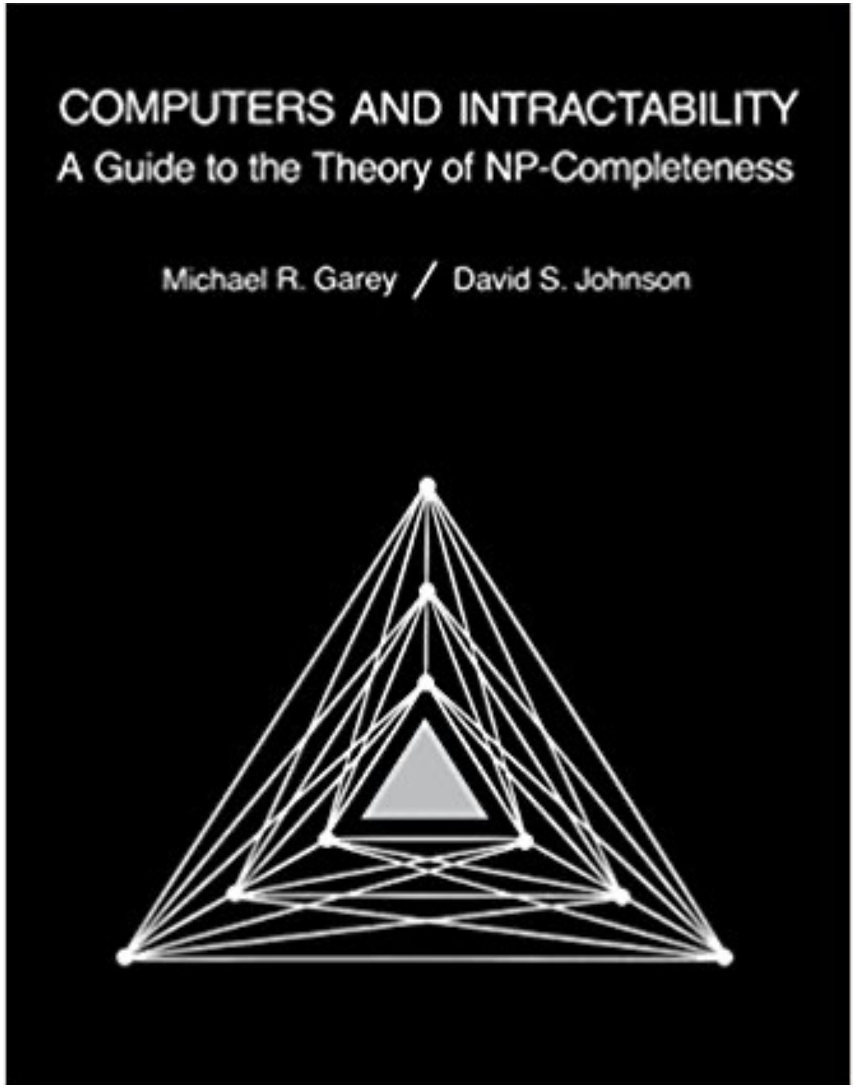
NP-Complete Problems

But Wait! There's more!

By 1979, at least 300 problems had been proven NP-complete.

Garey and Johnson put a list of all the NP-complete problems they could find in this textbook.

Took them almost 100 pages to just list them all.



What do we do about it?

- 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)